НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
Сергій СТІРЕНКО
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Метод підвищення якості відеозображення за допомогою нейронної мережі**»**

Виконав (-ла):
студент (-ка) VI курсу, групи ІВ-91мн
Прасолов Андрій Артурович _____

Керівник:
професор кафедри ОТ, д.ф.-м.н.
Гордієнко Юрій Григорович _____

Консультант з нормоконтролю:
професор кафедри ОТ, д.т.н.
Кулаков Юрій Олексійович _____

Рецензент:
доцент кафедри АУТС, к.т.н
Писаренко Андрій Володимирович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Спеціалізація *123. Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Стіренко С.Г.
(підпис) (ініціали, прізвище)
«_____» _____ 2021 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Прасолов Андрій Артурович
(прізвище, ім'я, по батькові)

1. Тема дисертації Метод підвищення якості відеозображення за допомогою нейронної мережі
Науковий керівник дисертації професор, д.ф.-м.н. Гордієнко Ю.Г.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «12» 03 2021 р. № 809-с

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження: метод підвищення якості відеозображення за допомогою нейронної мережі

4. Предмет дослідження: ефективність роботи методу покращення якості за допомогою нейронних мереж на відеозображеннях

5. Перелік завдань, які потрібно розробити: дослідити поширені методи покращення зображень та аналоги використовуваних в них нейромереж, спроектувати та розробити власний метод на основі дослідженого матеріалу, протестувати та отримати результат.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Кулаков Ю. О.		
1			
2			
3			
4			

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Затвердження теми роботи	17.09.2020	
2	Складання і узгодження технічного завдання	28.09.2020	
3	Написання вступної частини та огляд рішень	15.10.2020	
4	Розробка способу	22.12.2020	
5	Програмна реалізація	18.01.2021	
6	Оформлення пояснювальної записки	22.02.2021	
7	Попередній захист та проходження нормативного контролю		
8	Подання МД рецензенту		
9	Захист	17.05.2021	

Студент _____ Прасолов А. А.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Гордієнко Ю. Г.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему:

Метод підвищення якості відеозображення за допомогою нейронної мережі

студентом: Прасоловим Андрієм Артуровичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 75 аркушів основного тексту, 35 ілюстрацій, 3 таблиці. При підготовці використовувалася література з 63 різних джерел.

Актуальність. В останні роки все частіше для вирішення складних задач почали використовувати машинне навчання, та штучний інтелект у цілому. В 2017 році були продемонстровані екрани з роздільною здатністю 8К, тому питання покращення зображення та відео під надвисоку роздільну здатність є досить актуальним, оскільки вага таких зображень, та передача їх по мережі потребує достатньо велику кількість ресурсів. На допомогу у вирішенні таких задач існують нейромережі, які мають змогу з дуже низької роздільної здатності, отримати аналог зображення з роздільною здатністю, яка перевищує початкову більше ніж у 4 рази.

Особливо важливим аспектом у використанні методів покращення зображень за допомогою нейромереж, є їх адаптованість під обмежені ресурси, такі як використання мережі та обчислювальна здатність. Тому на сьогоднішній день дуже гостро ведуться змагання з оптимізованих методів покращення зображень та відео, що дають змогу навіть мобільним пристроям, витрачаючи мало ресурсів, за допомогою нейромереж отримувати високу якість.

Мета і завдання дослідження. Метою магістерської роботи є створення та розвиток способів поліпшення характеристик якості зображень на основі нейронної мережі, та дослідження можливостей їх застосування для покращення відео.

Для досягнення поставленої магістерською роботою мети було поставлено й вирішено наступні завдання:

- Дослідити базові архітектури нейромереж для покращення якості зображень та проблеми постановки питання у обробці таким методом;
- Порівняти та дослідити дискретизацію, види її обробки та методи підвищення;
- Розглянути типи нейромереж для покращення якості зображення, та їх властивості для використання у досліджених архітектурах;
- Проаналізувати оригінальний метод та розробити власний на основі використання більш сучасних нейромереж;
- Проілюструвати виконання обробки зображення реалізованим методом, та проаналізувати властивості для демонстрації переваг та недоліків у використанні його для обробки відеозображення;

Об’єкт дослідження. Метод підвищення якості відеозображення за допомогою нейронної мережі.

Предмет дослідження. Ефективність роботи методу покращення якості зображення та відео та шляхи його покращення.

Методи досліджень. Для досягнення поставлених в магістерській роботі задач, було використано методи машинного навчання та тренування нейронних мереж. Наукова новизна проведеного дослідження забезпечена наступними пунктами:

- Запропоновано новий блок з використанням більш сучасних та оптимізованих нейромереж для обробки зображень, а саме отримання більш реалістичного результату з використанням меншої кількості ресурсів.
- Реалізовано програмний продукт, що вирішує поставленні задачі для подальшого використання та досліджень.
- Тренування оригінального та реалізованих методів виконувалось на власному ПК за допомогою Anaconda та JupyterNotebook.

Особистий внесок здобувача. Магістерське дослідження є самостійно виконаною роботою, в якій відображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до вирішення задачі покращення якості зображення та відеозображення за

допомогою використання нейронних мереж. Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Практична цінність. Отримані результати можуть бути вільно використані у майбутніх дослідженнях за напрямками:

- Глибоке машинне навчання.
- Покращення якості для зображення та відеозображення нейронними мережами.
- Дослідження використання реалізованого методу на мобільних пристроях.

Публікації. Результати дослідження, яке було виконано в рамках дисертації, були представлені на наступних конференціях, та опубліковані в збірниках доповідей цих конференцій:

1. Prasolov, S. Stirenko, Y. Gordienko, Generative Adversarial Networks for Image Super Resolution, The International Conference on Security, Fault Tolerance, Intelligence (ICSFTI2021 Online), Kyiv, Ukraine.
2. Prasolov, S. Stirenko, Y. Gordienko, Improvement of Image Super Resolution by Deep Neural Networks, IEEE 19th International Conference on Smart Technologies (EUROCON-2021), Lviv, Ukraine.

Ключові слова. Згорткові нейронні мережі, Глибоке навчання, Генеративні змагальні мережі, Висока роздільна здатність, Низька роздільна здатність, Надвисока роздільна здатність.

Abstract

For a master's thesis

made on the topic: Method of Video Quality Improvement by Using a Neural Network

by student: Prasolov Andrii Arturovych

Master's Thesis: The study consists of an introduction and four sections. Total workload is: 75 sheets of body text, 35 illustrations, 3 tables. 63 various sources were used.

The urgency of the problem. In recent years, machine learning and artificial intelligence in general have been increasingly used to solve complex problems. Screens with 8K resolutions were demonstrated in 2017, so the issue of improving images and videos at ultra-high resolutions is quite relevant, as the weight of such images and their transmission over the network requires a large number of resources. To help solve such problems, there are neural networks that can, from a very low resolution, get an analog image with a resolution that exceeds the original more than 4 times.

A particularly important aspect in the use of methods to improve images using neural networks is their adaptability to limited resources, such as network use and computing power. Therefore, today there is a very fierce competition for optimized methods of image and video enhancement, which allow even mobile devices, spending little resources, to obtain high quality through neural networks.

The purpose and objectives of the study. The purpose of the masters research is to create and develop ways to improve the performance of image quality based on the neural network, and to explore the possibilities of their application to improve video.

To achieve the goal set by the master's thesis, the following tasks were set and solved:

- Investigate the basic architectures of neural networks to improve image quality and the problem of asking questions in processing by this method;
- Compare and investigate sampling, types of its processing and methods of increase;

- Consider the types of neural networks to improve image quality, and their properties for use in the studied architectures;
- Analyze the original method and develop your own based on the use of more modern neural networks;
- Illustrate the implementation of image processing by the implemented method, and analyze the properties to demonstrate the advantages and disadvantages of using it for video processing;

Object of the study. Method of Video Quality Improvement based on a Neural Network.

Subject of study. The effectiveness of the method of improving the quality of images and videos and ways to improve it.

Research methods and Scientific Novelty. To achieve the objectives set in the master's thesis, the methods of machine learning and training of neural networks were used. The scientific novelty of the study is provided by the following points:

- A new unit is proposed using more modern and optimized neural networks for image processing, namely to obtain a more realistic result using fewer resources.
- Implemented a software product that solves the problem for further use and research.
- Training of original and implemented methods was performed on your own PC using Anaconda and JupyterNotebook.

Personal contribution of the applicant. The master's research is a self-performed work, which reflects the personal author's approach and personally obtained theoretical and applied results related to solving the problem of improving the quality of images and video images through the use of neural networks. The formulation of the purpose and objectives of the study was carried out jointly with the supervisor.

Practical value. The obtained results can be freely used in future research in the following areas:

- Deep machine learning.
- Improved image and video quality for neural networks.
- Study of the use of the implemented method on mobile devices.

Publications. The results of the research, which was performed as part of the dissertation, were presented at the following conferences, and published in the proceedings of these conferences:

1. A. Prasolov, S. Stirenko, Y. Gordienko, Generative Adversarial Networks for Image Super Resolution, The International Conference on Security, Fault Tolerance, Intelligence (ICSFTI2021 Online), Kyiv, Ukraine.
2. A. Prasolov, S. Stirenko, Y. Gordienko, Improvement of Image Super Resolution by Deep Neural Networks, IEEE 19th International Conference on Smart Technologies (EUROCON-2021), Lviv, Ukraine.

Keywords. Convolutional neural networks (CNNs), Deep learning, Generative adversarial networks (GANs), High-resolution (HR), Low resolution (LR), Super-resolution (SR).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ANR (англ. accelerated neuroregulation) - прискорена нейрорегуляція

BCI (англ. bicubic interpolation) - бікубічна інтерполяція

BLI (англ. bilinear interpolation) - білінійна інтерполяція

BN (англ. batch normalization) - пакетна нормалізація

BSDS (англ. berkeley segmentation dataset) - набір даних сегментації Берклі

CLI (англ. command line interface) - інтерфейс командного рядка

CNN (англ. convolutional neural network) - згорткова нейронна мережа

CORNIA (англ. codebook representation for no-reference image assessment) - подання кодової книги для оцінки зображення без посилань

CPU (англ. central processing unit) - центральний процесор

CUDA (англ. compute unified device architecture) - уніфікована архітектура обчислювального пристрою

CVPR (англ. conference on computer vision and pattern recognition) - конференція з комп'ютерного зору та розпізнавання образів

CelebA (англ. celebrity faces attribute dataset) - набір даних атрибутів знаменитостей

DBPN (англ. deep back-projection networks) - глибокі мережі зворотного проектування

ECCV (англ. european conference on computer vision) - європейська конференція з комп'ютерного зору

EDSR (англ. enhanced deep residual networks for single image super-resolution) - вдосконалені глибокі залишкові мережі для надвисокої роздільної здатності одного зображення

ESPCN (англ. efficient sub-pixel convolutional neural network) - ефективна субпіксельна згорткова нейронна мережа

FSIM (англ. feature similarity index for image quality assessment) - індекс подібності ознак для оцінки якості зображення

FSRCNN (англ. fast convolutional neural network for image super-resolution) - швидка згорткова нейронна мережа для надвисокої роздільної здатності зображення

GAN (англ. generative adversarial network) - генеративна змагальна мережа

GPU (англ. graphical processing unit) - графічний процесор

GUI (англ. graphical user interface) - графічний інтерфейс користувача

HR (англ. high resolution) - висока роздільна здатність

HVS (англ. human visual system) - зорова система людини

HTML (англ. hypertext markup language) - мова розмітки гіпертексту

JSON (англ. JavaScript object notation) - позначення об’єкта JavaScript для текстового обміну даними

ILSVRC (англ. ImageNet large scale visual recognition challenge) - масштабне змагання розпізнавання зображень ImageNet

IQA (англ. image quality assessment) - оцінка якості зображення

LR (англ. low resolution) - низька роздільна здатність

LapSRN (англ. deep Laplacian pyramid networks for fast and accurate super-resolution) - глибокі лапласівські пірамідні мережі для швидкої та точної надвисокої роздільної здатності

MAC (англ. multiply-accumulate) - помножити-додати

MBConv (англ. inverted residual block) - перевернутий залишковий блок

MDSR (англ. multi-scale deep super-resolution system) - багатомасштабна система глибокої надвисокої роздільної здатності

MOS (англ. mean opinion score) - середня оцінка думки

MS-COCO (англ. microsoft common objects in context dataset) - набір даних загальних контекстних об’єктів Microsoft

MS-LapSRN (англ. multiscale Laplacian pyramid super-resolution network) - багатомасштабна мережа надвисокої роздільної здатності піраміди Лапласа

MSE (англ. mean squared error) - середньоквадратична помилка

MS-SSIM (англ. multiscale structural similarity) - багатомасштабна структурна подібність

MemNet (англ. deep persistent memory network) - мережа глибокої стійкої пам'яті

NIQE (англ. naturalness image quality evaluator) - оцінювач якості природності зображення

NTIRE (англ. new trends in image restoration and enhancement workshop) - семінар з тенденцій у відновленні та вдосконаленні зображень

PDF (англ. portable document format) - формат портативного документа

PIRM (англ. perceptual image restoration and manipulation workshop) - семінар з перцептивного відновлення та маніпулювання зображеннями

PNG (англ. portable network graphics) - портативна мережева графіка

PSNR (англ. peak signal-to-noise ratio) - пікове відношення сигнал / шум

ProSR (англ. progressive approach to single-image super-resolution) - прогресивний підхід до надвисокої роздільної здатності одного зображення

RAM (англ. random access memory) - оперативна пам’ять

RBPN (англ. recurrent back-projection network for video super-resolution) - рекурентна мережа зворотної проекції для надвисокої роздільної здатності відео

RDN (англ. residual dense network for image super-resolution) - залишкова щільна мережа для надвисокої роздільної здатності зображення

REPL (англ. read-eval-print loop) - цикл читання-обчислення-друку

RGB (англ. red green blue) - червоний, зелений та синій кольори

RMSE (англ. root mean square error) - середньоквадратична помилка

ReLU (англ. rectified linear unit) - випрямлена лінійна одиниця

ResNet (англ. residual network) - залишкова мережа

SISR (англ. single image super-resolution) - надвисока роздільна здатність одного зображення

SR (англ. super-resolution) - надвисока роздільна здатність

SRCNN (англ. super-resolution convolutional neural network) - згорткова нейронна мережа для надвисокої роздільної здатності

SRDenseNet (англ. super-resolution dense network) - щільна мережа для надвисокої роздільної здатності

SRFBN (англ. feedback network for image super-resolution) - мережа зворотного зв'язку для надвисокої роздільної здатності зображення

SRGAN (англ. generative adversarial network for image super-resolution) - генеративна змагальна мережа для надвисокої роздільної здатності зображення

SRResNet (англ. residual network for super-resolution) - залишкова мережа для надвисокої роздільної здатності

SSIM (англ. structural similarity index) - індекс структурної подібності

TPU (англ. tensor processing unit)

WDSR (англ. wide activation for efficient and accurate image super resolution) широка активація для ефективної та точної надвисокої роздільної здатності зображення

ВСТУП

В останні роки ми стали свідками значного прогресу в області надвисокої роздільної здатності зображень із використанням методів глибокого машинного навчання. Надвисока роздільна здатність (Super resolution, SR) - це важливий клас методів обробки зображень для підвищення роздільної здатності зображень і відео в комп'ютерному баченні. Дана робота демонструє дослідження останніх досягнень у цій області із використанням підходів глибокого навчання. В цій області ми можемо грубо згрупувати існуючі дослідження методів SR за трьома основним категоріями: контрольована, неконтрольована та домен-специфічна SR.

Метод SR, який відноситься до процесу відновлення зображень з високою роздільною здатністю (High resolution, HR) із зображень з низькою (Low Resolution, LR), є важливим класом зображень у методах обробки зображень в комп'ютерному баченні [1]. Він має широкий спектр реальних додатків, такі як медична візуалізація [2], спостереження та безпека та інші [3]. Крім покращення якості сприйняття, цей метод також допомагає вирішити інші завдання комп’ютерного бачення [4, 5]. Загалом, ця проблема дуже складна і по своїй суті некоректна, оскільки завжди є кілька результуючих зображень HR, що відповідають одному зображенню LR. У літературі описані різні класичні методи HR, в тому числі методи, засновані на прогнозуванні [6], крайові методи [7], статистичні методи [8], методи-патчі [9] та розріджені методи представлення [10].

З швидким розвитком методів глибокого навчання в останні роки SR-моделі, засновані на глибокому навчанні, активно досліджувалися і часто досягали високого рівня продуктивності в різних тестах. Різноманітність глибоких методів навчання застосовувалися для вирішення завдань SR, починаючи з ранніх згорткових нейронних мереж (CNN) заснованих на методі, наприклад, SRCNN [11], та багатообіцяючих підходів SR з використанням генеративних змагальних мереж (GAN), наприклад, метод фото-реалістичної надвисокої роздільної здатності одного зображення за допомогою генеративної змагальної мережі [12] (SRGAN) [13]. В цілому, сімейство алгоритмів SR, що

використовують методи глибокого навчання, відрізняються один від одного в наступних основних аспектах: різні типи мережевої архітектури, різні типи втрат функцій, різні типи принципів навчання і стратегії і т. д [14].

Тому, тема магістерської роботи, а саме дослідження існуючих методів SR з використанням глибокого машинного навчання є надзвичайно актуальною. Та спрямована на покращення результату виконання обробки зображень, а саме на використання більш сучасних та розповсюджених видів нейронних мереж.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ ВИДІВ ОБРОБКИ ЗОБРАЖЕНЬ ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОТРИМАННЯ НАДВИСОКОЇ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕННЯ

Основний аналіз методів складається з наступних кроків [1]:

- Огляд методів SR зображень, заснованих на глибокому навчанні, включно набори завдань, набори тестових даних та показники продуктивності.
- Систематичний огляд останніх досягнень методів SR на основі глибокого навчання ієрархічним і структурним чином, та дослідження переваг і обмежень кожного компоненту для ефективного вирішення проблеми SR.

Ми розглянемо різні аспекти останніх досягнень в області надвисокої роздільної здатності зображень з глибоким навчанням. На рис.1.1. показана ієрархія структури методу та особливостей SR.

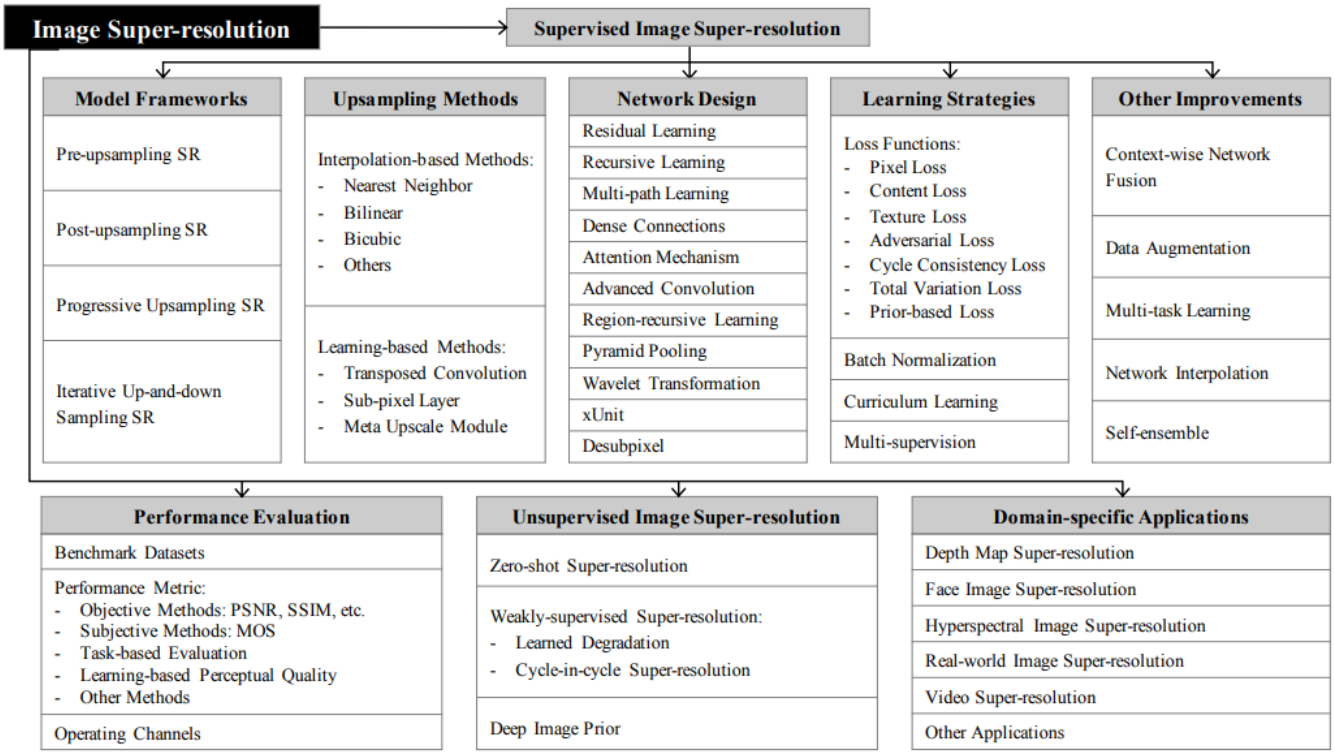


Рис. 1.1. Ієрархічна структура методу SR [1]

1.1. Огляд підходу для отримання надвисокої роздільної здатності зображення

SR зображення направлено на відновлення відповідних зображень HR з зображень LR. Як правило, зображення з низькою роздільною здатністю моделюється як результат наступної деградації:

$$I_x = \mathcal{D}(I_y; \delta), \qquad (1)$$

де *D* позначає функцію відображення деградації, I_y - відповідне зображення HR, а δ – параметри процесу погіршення (наприклад, коефіцієнт масштабу або шум). Як правило, процес деградації (тобто *D* і δ) невідомий, і надаються тільки зображення LR. У цьому випадку також відомий як сліпа SR, дослідники повинні відновити HR - апроксимацію Iy правдивого HR зображення Iy з
зображення LR Ix, як показано далі:

$$\hat{I}_y = \mathcal{F}(I_x; \theta), \qquad (2)$$

де *F* - модель надвисокої роздільної здатності, а θ - параметри *F*. Хоча процес деградації невідомий і на нього можуть впливати різні чинники (наприклад, артефакти стиснення, анізотропні деградації, шум сенсора і т.д.), дослідники намагаються змоделювати відображення деградації. Більшість результуючих робіт безпосередньо моделюють погіршення як одну операцію понижувальної дискретизації, як показано нижче:

$$\mathcal{D}(I_y; \delta) = (I_y) \downarrow_s, \{s\} \subset \delta, \qquad (3)$$

де $\downarrow_s$ - операція зниження дискретизації з масштабуванням
фактору *s*. Фактично, більшість наборів даних для загального SR побудовані на основі цього шаблону, і найбільш часто використовувана операція понижувальної дискретизації - це бікубічна інтерполяція зі згладжуванням. Однак є й інші роботи [39], що моделюють
деградацію, як поєднання декількох операцій:

$$\mathcal{D}(I_y; \delta) = (I_y \otimes \kappa) \downarrow_s + n_\varsigma, \{\kappa, s, \varsigma\} \subset \delta, \qquad (4)$$

де *Iy* $\otimes$ κ представляє згортку між розмиванням ядра κ і зображення з HR *Iy*, а n_ς - це якась додаток до чистого шуму Гауса із стандартним відхиленням. В

порівнянні до визначенням рівняння (3), деградація рівняння (4) ближче до реальних випадків і було продемонстровано [15], що вона більш корисна для SR. Отже, мета SR полягає в наступному:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\hat{I}_y, I_y) + \lambda\Phi(\theta), \qquad (5)$$

де *L(ˆIy, Iy)* представляє собою функцію втрат між згенерованим зображенням з HR *ˆIy* та базовим істинним зображенням *Iy*, *Φ(θ)* - член врегулювання, а λ - параметр компромісу [1].

Не дивлячись на те, що сама популярна функція втрат для SR - піксельна середньоквадратична помилка (тобто втрата пікселів), більш потужні моделі схильні використовувати комбінацію декількох функцій втрат, які
буде розглянуто далі.

1.2. Набори даних для методу отримання надвисокої роздільної здатності

На сьогоднішній день для зображень доступні найрізноманітніші набори даних для надвисокої роздільної здатності зображень, які значно відрізняються по кількості зображень, якості, роздільній здатності, різноманітності і т. д. Деякі з них пропонують пари зображень HR-LR, в той час як інші пропонують лише зображення HR, у цьому випадку зображення LR, як правило, отримують функцію *imresize* із стандартними налаштуваннями у MATLAB (тобто бікубічна інтерполяція зі згладжуванням)[1]. У таблиці 1.1 представлено основні дані для порівняння різних наборів даних, у тому числі кількість зображень HR, середню роздільну здатність зображень, формат зображень і категорії.
Крім цих наборів даних, деякі з них, широко використовуються для
інших завдань спостереження для SR, такі як ImageNet [16], MS-COCO [17], VOC2012 [18], CelebA [19]. Об'єднання кількох наборів даних для навчання також
популярні, такі як T91 і BSDS300 [20], об'ємна DIV2K і Flickr2K [21].

Таблиця 1.1.

Список наборів зображень для SR з відкритим доступом

Набір даних	Кількість зображень	Середня роздільна здатність	Середня к-сть пікселів	Формат	Ключові слова категорії
BSDS300	300	(435, 367)	154 401	JPG	тварина, будівля, їжа, простір, людина, рослина тощо.
BSDS500	500	(432, 370)	154 401	JPG	тварина, будівля, їжа, простір, люди, рослина тощо.
DIV2K	1000	(1972, 1437)	2 793 250	PNG	довкілля, флора, фауна, рукотворний об’єкт, люди, декорації тощо.
General-100	100	(435, 381)	181 108	BMP	тварина, щоденна необхідність, їжа, люди, рослина, текстура тощо.
L20	20	(3843, 2870)	11 577 492	PNG	тварина, будівля, краєвид, люди, рослина тощо.
Manga109	109	(826, 1169)	966 011	PNG	обсяг manga
OutdoorScene	10624	(553, 440)	249 593	PNG	тварина, будівля, трава, гора, завод, небо, вода
PIRM	200	(617, 482)	292 021	PNG	середовища, флора, природні пейзажі, предмети, люди тощо.
Set5	4	(313, 336)	113 491	PNG	дитина, птах, метелик, голова, жінка
Set14	14	(492, 446)	230 203	PNG	люди, тварини, комахи, квіти, овочі, комікси, гірки тощо.
T91	91	(264, 204)	58 853	PNG	автомобіль, квітка, фрукти, людське обличчя тощо.
Urban100	100	(984, 797)	774 314	PNG	архітектура, місто, структура, міський тощо.

1.3. Методи оцінки якості зображення

Якість зображення відноситься до візуальних характеристик зображень і фокусується на оцінках сприйняття глядачів. В цілому, методи оцінки якості зображення (IQA) включають суб'єктивні методи, засновані на людському сприйнятті (тобто наскільки реалістично зображення виглядає) і об'єктивні обчислювальні методи.

Перший варіант більше відповідає нашим потребам, але часто вимагає багато часу, та затрат, тому другий варіант є більш пріоритетним. Однак ці методи не обов'язково узгоджені між собою, тому що об'єктивні методи часто не здатні вловити зорове сприйняття людини дуже точно, що може привести до великої різниці в результатах IQA [13, 22].

Крім того, об'єктивні методи IQA діляться на три типи: повні еталонні методи, які виконують оцінку з використанням еталонних зображень, скорочені еталонні методи, засновані на порівнянні витягнутих ознак, і методи без зв’язку (тобто сліпий IQA) без будь-яких еталонних зображень. Найбільш часто використовувані методи IQA, що охоплюють як суб'єктивні методи, так і об'єктивні методи.

Пікове відношення сигнал / шум (PSNR) - одне з найбільш популярних оцінювань якості реконструкції перетворення з втратами (наприклад, стиснення, перемальовування зображення) для зображення HR, PSNR визначається через максимальне значення пікселів (позначається L) і середньоквадратичну помилку (MSE) між зображеннями. З огляду на вихідне зображення I з N пікселів і реконструкція I, PSNR між I і $\hat{I}$ визначаються наступним чином:

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{L^2}{\frac{1}{N}\sum_{i=1}^{N}(I(i) - \hat{I}(i))^2}\right), \qquad (6)$$

де L дорівнює 255 в загальних випадках з використанням 8-бітних репрезентацій. Оскільки PSNR відноситься тільки до піксельного рівня MSE, піклуючись тільки про відмінності між відповідними пікселями, а не про візуальне сприйняття, це часто призводить до поганої продуктивності в поданні якості реконструкції в

реальні сцени, де ми зазвичай більше стурбовані людським сприйняттям. Однак у зв'язку з необхідністю порівняння з художніми витворами і відсутністю повністю точних метрик сприйняття, PSNR в даний час залишається найбільш широко використовуваним критерієм оцінки для моделей SR.

Структурна подібність - з огляду на те, що здорова система людини (HVS) надзвичайно адаптована для вилучення структур зображень, структурний індекс подібності (SSIM) [22] пропонується для вимірювання структурної подібності між зображеннями, заснованої на незалежних порівняннях за яскравістю, контрастом і структурі. Для зображення *I* з *N* пікселями яскравість *μI* і контраст *σI* оцінюються як середнє і стандартне відхилення інтенсивності зображення відповідно, тобто

$$\mu_I = \frac{1}{N}\sum_{i=1}^{N} I(i)$$

$$\sigma_I = \left(\frac{1}{N-1}\sum_{i=1}^{N}(I(i)-\mu_I)^2\right)^{\frac{1}{2}}$$

де *I(i)* це яскравість i-го пікселя зображення *I*. Порівняння від яскравості і контрасту, що позначаються як *Cl(I, ˆI) і Cc(I, ˆI)* відповідно, даються за формулою:

$$\mathcal{C}_l(I,\hat{I}) = \frac{2\mu_I\mu_{\hat{I}} + C_1}{\mu_I^2 + \mu_{\hat{I}}^2 + C_1}, \qquad (7)$$

$$\mathcal{C}_c(I,\hat{I}) = \frac{2\sigma_I\sigma_{\hat{I}} + C_2}{\sigma_I^2 + \sigma_{\hat{I}}^2 + C_2}, \qquad (8)$$

де $C1 = (k1L)^2$ і $C2 = (k2L)^2$ є константами для уникнення нестабільності, *k1* і *k2*. Крім того, структура зображення представлена нормалізованими значеннями пікселів (тобто *(I - μI) / σI*), кореляції яких вимірюють структурну подібність, еквівалентно до коефіцієнта кореляції між *I* і *I*. Таким чином функція порівняння структур *Cs(I, ˆI)* визначається як:

$$\sigma_{I\hat{I}} = \frac{1}{N-1}\sum_{i=1}^{N}(I(i)-\mu_I)(\hat{I}(i)-\mu_{\hat{I}}), \qquad (9)$$

$$\mathcal{C}_s(I,\hat{I}) = \frac{\sigma_{I\hat{I}} + C_3}{\sigma_I\sigma_{\hat{I}} + C_3}, \qquad (10)$$

де *σI, I* кореляція між *I* та *I*, а *C3* постійна для стабільності.

Отже, SSIM визначається:

$$\mathrm{SSIM}(I,\hat{I}) = [\mathcal{C}_l(I,\hat{I})]^{\alpha}[\mathcal{C}_c(I,\hat{I})]^{\beta}[\mathcal{C}_s(I,\hat{I})]^{\gamma}, \qquad (11)$$

де α, β, γ - керуючі параметри для налаштування відносної важливості. Оскільки SSIM оцінює якість реконструкції з точки зору HVS, він краще відповідає вимогам оцінки сприйняття, а також широко використовується [23].

Середня оцінка думки (MOS) Тестування середньої оцінки думки (MOS) є широко використовуваним суб'єктивним методом IQA, при якому людей-оцінювачів просять присвоїти оцінки якості сприйняття оцінюваних зображень. Зазвичай оцінка становить від 1 (погано) до 5 (добре). І остаточний MOS розраховується як середнє арифметичне за всіма рейтингами. Хоча тестування MOS здається вірним методом IQA, йому притаманні деякі недоліки, такі як нелінійне сприйняття шкали, зміщення і дисперсія критеріїв оцінки. Насправді, є деякі моделі SR, які погано працюють із загальними показниками IQA (наприклад, PSNR), але набагато перевершують інші з точки зору якості сприйняття, і в цьому випадку тестування MOS є найбільш надійним методом IQA для точного вимірювання якості сприйняття [13], [14].

Якість сприйняття на основі навчання – для того щоб краще оцінити якість сприйняття зображення при одночасному скороченні ручного втручання, дослідники намагаються оцінити якість сприйняття шляхом навчання на великих наборах даних. Хоча ці методи демонструють кращу продуктивність при уловлюванні візуального сприйняття людини, яка якість сприйняття нам потрібна (наприклад, більш реалістичні зображення або узгоджена ідентичність вихідного зображення) залишається питанням, що вимагає вивчення, тому об'єктивні методи IQA (наприклад, PSNR, SSIM) в даний час залишаються головними [24].

Оцінка на основі завдань- так як моделі SR часто можуть допомогти іншим завданням для бачення, оцінка ефективності реконструкції за допомогою інших завдань - ще один ефективний спосіб [4], [5], [14]. Зокрема, дослідники завантажують вихідні та відновлені зображення HR в навчені моделі і оцінюють

якість реконструкції шляхом порівняння впливу на продуктивність прогнозування. Завдання бачення, використовують для оцінки розпізнавання об'єктів [14], розпізнавання осіб [25], вирівнювання і аналіз осіб і т. д.

Інші методи IQA - крім перерахованих вище методів IQA, існують інші менш популярні метрики SR. Різномасштабні структурні подібності (MS-SSIM) [26] забезпечують більшу гнучкість, ніж однорівневий SSIM з урахуванням варіацій умов перегляду. Функція подібності ознак (FSIM) [27] витягує характерні точки людського інтересу на основі збігу фаз і зображення величини градієнта для оцінки якості зображення. Природний оцінювач якості зображення (NIQE) [28] використовує вимірні відхилення від статистичних закономірностей, які спостерігаються в природних зображеннях, без впливу на спотворені зображення.

1.4. Канали для оперування зображенням

Додатково до зазвичай використовуваного кольорового простору RGB, кольорова палітра YCbCr також широко використовується для SR. У цьому просторі зображення представлені каналами Y, Cb, Cr, що позначають яскравість, кольоровість синього і червоного кольорів компонентів відповідно.

Хоча в даний час немає прийнятого передового досвіду для виконання або оцінки простору надвисокої роздільної здатності, більш ранні моделі віддають перевагу роботі на каналі Y простору YCbCr [29], а пізніші моделі, як правило, працюють з каналами RGB [21]. Варто відзначити, що операції (тренування або оцінка) на різних колірних просторах або каналах може призвести до сильного розрізнення оцінки (до 4 дБ) [30].

1.5. Дослідження проблем з надвисокою роздільною здатністю зображення

Завдання «Нові тенденції у відновленні і поліпшенні зображення» (NTIRE) пов'язана з CVPR і включає в себе кілька завдань, таких як SR, шумозаглушення і розфарбовування. Для зображень з SR завдання NTIRE [31] побудовано на наборі даних DIV2K [32] і складається з бікубічного зменшення розмірів доріжок

і сліпих доріжок з реалістичною невідомою деградацією. Ці типи розрізняються за ступенем деградації і коефіцієнтам масштабування та спрямовані на просування досліджень SR в обох випадках, ідеальні умови і реальні несприятливі ситуації.

Перцептивне відновлення зображення та маніпуляції (PIRM) [33] пов'язане з

ECCV, а також включає в себе кілька завдань. На відміну від NTIRE, одна з під задач PIRM фокусується на компромісі між точністю генерації і якістю сприйняття, а інший присвячений SR на смартфонах. Як добре відомо, моделі, націлені на спотворення, часто дають візуально неприємні результати, в той час як моделі націлені на якість сприйняття погано показують себе на достовірності інформації. Зокрема, PIRM розділив сприйняття площини на три області відповідно до порогами на корені середньоквадратичної помилки (RMSE). У кожній області виграшний алгоритм - це той, який забезпечує найкращу якість сприйняття. У той час як в інша під задача, SR на смартфонах, де учасники просять отримати SR зображення з обмеженим використанням смартфону (Включаючи CPU, GPU, RAM і т. д.), А оціночні показники включають тестування PSNR, MS-SSIM і MOS. Таким чином,
PIRM заохочує передові дослідження компромісу спотворення сприйняття, а також сприяє легкості та ефективності поліпшення зображення на смартфонах.

1.6. Контрольоване отримання зображення з надвисокою роздільною здатністю

В даний час дослідники запропонували безліч моделей SR з глибоким навчанням. Ці моделі орієнтовані на контрольований SR, тобто навчений з наборами зображень HR та відповідними наборами зображень HR. Хоча відмінності між цими моделями дуже великі, деякі комбінації набору компонентів, таких як модель фреймворку, методи попередньої дискретизації, дизайн мережі та вивчаючи стратегії є схожими. З цієї точки зору дослідники комбінують ці компоненти, щоб побудувати інтегровану модель SR для підходу до конкретних цілей.

В отриманні надвисокої роздільної здатності зображення нас найбільш за все цікавить виконання попередньої дискретизації (тобто генерації вихідного сигналу HR з введення LR). Хоча архітектури існуючих моделей сильно відрізняються, їх можна віднести до чотирьох модельних каркасів (як показано на Рис. 2), що засновані на використовуваних операціях попередньої дискретизації та їх розташування в моделі.

1.6.1. Метод попередньої дискретизації

Через складність безпосереднього вивчення картографії від маловимірних до багатовимірних просторів, використання традиційних алгоритмів підвищує дискретизацію для отримання зображень з більш високою роздільною здатністю, а потім простим рішенням буде покращити їх за допомогою глибоких нейронних мереж (рис 1.2).

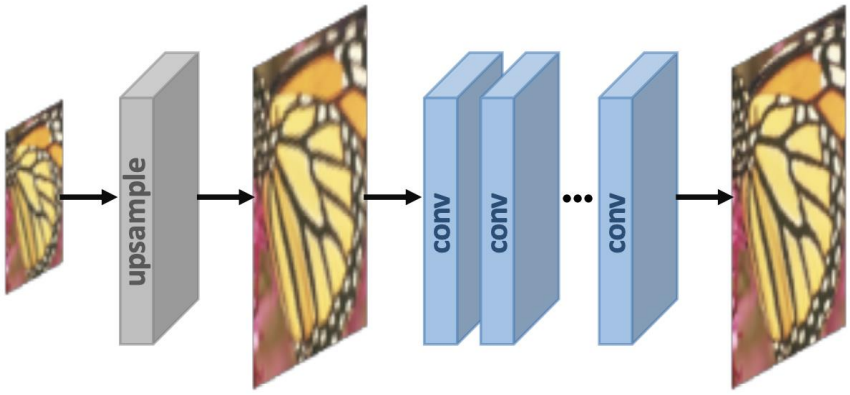


Рис. 1.2. Метод попередньої дискретизації [1]

Таким чином спочатку застосовують структуру SR для попередньої дискретизації, далі потрібно використати SRCNN, щоб вивчити відображення від кінця до кінця з інтерпольованих зображень LR в зображення HR. Зокрема, зображення LR підвищуються до поганої якості HR зображення бажаного розміру з використанням традиційних методів (наприклад, бікубічна інтерполяція).

Оскільки найскладніша операція з підвищення дискретизації була завершена, для відновлення якісних деталей CNN потрібно тільки уточнити грубі зображення, що значно знижує складність навчання. Крім того, ці моделі можуть робити інтерпольовані зображення з довільними розмірами і

коефіцієнтами масштабування в якості вхідних даних, тоді ми отримуємо поліпшені результати з порівнянною продуктивністю з однорівневими моделями SR [34]. Таким чином, поступово він став одним з найбільш популярних фреймворків [20], а також основні відмінності між цими моделями - це апостеріорна модель дизайну і стратегії навчання. Тим не менш, збільшення дискретизації часто викликає побічні ефекти (наприклад, посилення шуму і розмиття), і оскільки більшість операцій виконуються в багатовимірному просторі, вартість часу на простір набагато вище, ніж у інших фреймворків.

1.6.2. Метод пост-дискретизації

Щоб підвищити обчислювальну ефективність і зробити повне використання технології глибокого навчання для збільшення роздільної здатності автоматично, дослідники пропонують виконувати більшу частину обчислень в маловимірному просторі, замінюючи зумовленим підвищенням дискретизації з наскрізними шарами - учнями, інтегрованими в кінець моделей.

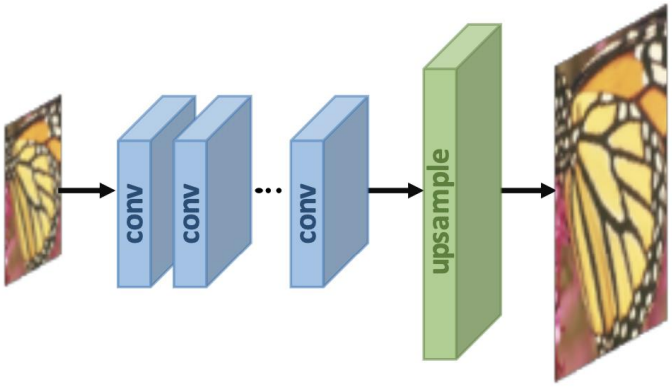


Рис. 1.3. Метод пост-дискретизації [1]

SR після підвищення частоти дискретизації, як показано на рис. 1.3, що вхідні зображення високої роздільної здатності подаються в глибокі CNN без збільшення роздільної здатності та повністю навчені шари дискретизації додаються в кінці мережі [29]. Оскільки процес вилучення ознак з величезними обчислювальними витратами відбувається тільки в маловимірних просторах і дозвіл збільшується тільки в кінці, обчислення та просторова складність значно зменшена. Отже, цей фреймворк також став одним з найпопулярніших [31]. Ці

моделі відрізняються в основному в навчальних шарах дискретизації, CNN структурами та стратегіями навчання.

1.6.3. Метод прогресивної дискретизації

Хоча фреймворк для отримання надвисокої роздільної здатності після підвищення дискретизації надзвичайно зменшив обчислювальну вартість, але є і недоліки. З одного боку, дискретизація виконується всього за один крок, що значно збільшує складність навчання для великих коефіцієнтів масштабування. З іншого боку, кожен коефіцієнт масштабування вимагає навчання індивідуальної моделі SR, яка не справляється з необхідністю багато масштабованої SR. Щоб усунути ці недоліки, структура прогресивного збільшення дискретизації адаптована мережею SR Лапласівської піраміди (LapSRN) [35], як показано на рис. 1.4.

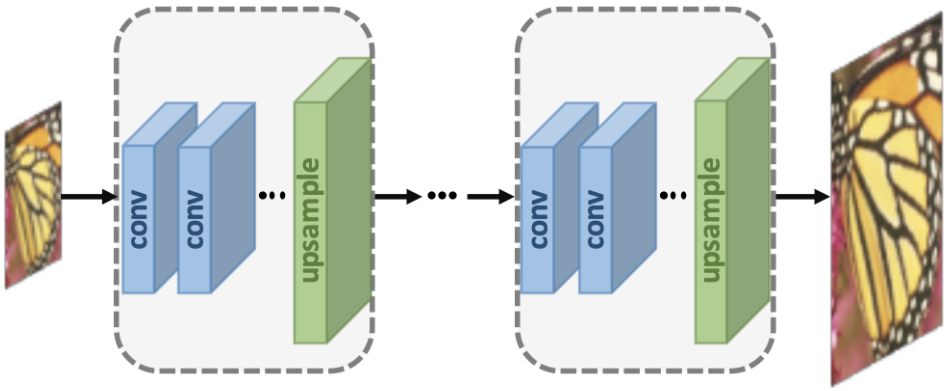


Рис. 1.4. Метод прогресивної дискретизації [1]

Зокрема, моделі в рамках цієї структури засновані на каскаді CNN і поступово відновлюють зображення з більш високою роздільною здатністю. На кожному етапі, зображення підвищуються до більш високої роздільної здатності та уточнюється CNN. Інші роботи, такі як MS-LapSRN [36] і прогресивні SR (ProSR) [37] також приймають цю структуру і досягають високої продуктивності. На відміну від LapSRN і MS-LapSRN з використанням проміжних реконструйованих зображень в якості «базових зображень» для наступних модулів, ProSR зберігає основний потік інформації і реконструює зображення середньої роздільної здатності по окремим розділам.

Розклавши складну задачу на прості завдання, моделі в рамках цієї структури значно скорочують труднощі навчання, а також можуть впоратися з багатомасштабованою SR без введення надмірно просторової і тимчасової вартості. Крім того, деякі специфічні стратегії навчання, такі як навчання за навчальною програмою, можуть бути безпосередньо інтегровані для подальшого зменшення складності навчання і поліпшення кінцевої продуктивності. Однак ці моделі також стикаються з деякими проблемами, такими як складне моделювання, багатоетапне проектування, стабільність тренування та ін.

1.6.4. Метод ітеративного підвищення та зниження дискретизації

Щоб краще відстежити взаємозалежність пар зображень LR, HR, ефективна ітераційна процедура, названа зворотною проекцією. Ця структура SR, а саме ітеративне підвищення і зниження дискретизації, намагається ітеративно застосувати зворотню проекцію уточнення, тобто обчислення помилки реконструкції з подальшим її об'єднанням для налаштування інтенсивності зображення HR. Загалом використовують ітераційні шари вибірки DBPN [38], який поперемінно з'єднує рівні підвищеної і зниженої дискретизації та відновлює остаточний результат HR використовуючи всі проміжні реконструкції. По аналогії, SRFBN використовує ітеративну висхідну і спадну вибірку блоку зворотного зв'язку з більш щільним пропуском підключень і кращим навчанням. RBPN [39] для відео з SR витягує контекст з безперервних відеокадрів і об'єднує цей контекст для створення повторюваного виведення кадрів модулем зворотної проекції.

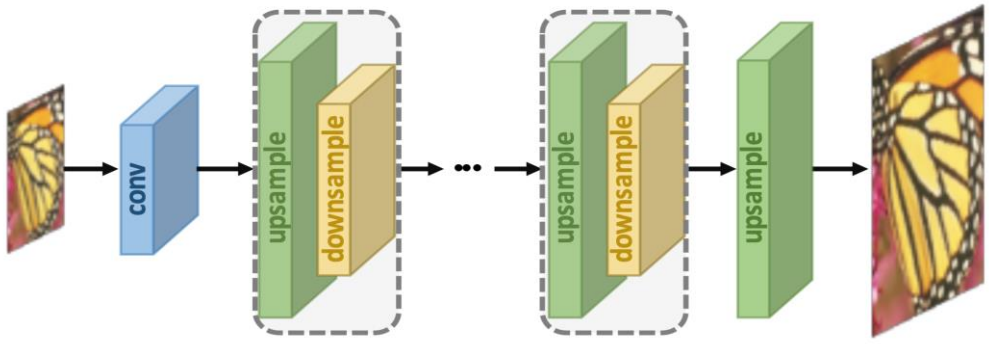


Рис. 1.5. Метод ітеративного підвищення та зниження дискретизації [1]

Моделі в рамках цієї структури можуть краще добувати глибокі зв’язки між парами зображень HR-LR і, отже, забезпечити більш якісні результати реконструкції. Проте, критерії проектування модулів зворотної проекції як і раніше не визначені.

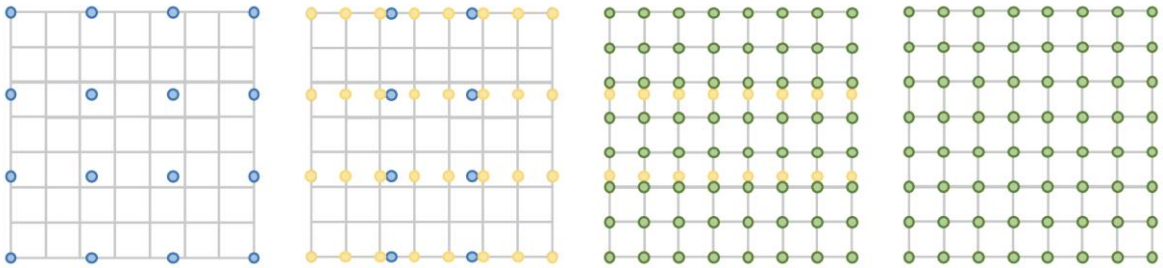
1.7. Методи підвищення дискретизації

Додатково до позицій підвищення дискретизації в моделі, велике значення має сам процес її виконання. Незважаючи на те, що існували різні традиційні методи підвищення дискретизації, використання CNN для вивчення їх наскрізної роботи поступово стало тенденцією.

1.7.1. Метод підвищення дискретизації на основі інтерполяції

Інтерполяція зображення, тобто масштабування зображення, відноситься до зміни розміру цифрових зображень і широко використовується програмами, пов’язаними із зображеннями. Традиційні методи інтерполяції включають в себе інтерполяцію найближчого сусідства, білінійну та бікубічну інтерполяцію, дискретизацію Синка та Ланцоша та ін. Оскільки ці методи є зрозумілі та легкі у реалізації, деякі з них все ще широко використовуються в моделях SR на базі CNN. Інтерполяція найближчого сусіда.

Інтерполяція найближчого сусідства - це простий та інтуїтивно зрозумілий алгоритм що відбирає значення найближчого пікселя для кожної позиції, яку потрібно інтерполювати, незалежно від будь-яких інших пікселів. Таким чином, цей метод є дуже швидким, але зазвичай дає блоковані результати низької якості.



(а) Початок (б) Крок 1 (в) Крок 2 Кінець

Рис. 1.6. Методи підвищення вибірки на основі інтерполяції. Сіра сітка позначає координати пікселів, сині, жовті та зелені точки представляють початковий, проміжний та вихідний пікселі відповідно [1]

Білінійна інтерполяція (BLI) спочатку виконує лінійну інтерполяцію на одній осі зображення, а потім виконує на іншій осі, як показано на рис. 1.6. Оскільки це призводить до квадратичної інтерполяції з рецептивним полем розміром 2 × 2, він показує набагато кращу продуктивність, ніж інтерполяція найближчого сусіда, зберігаючи відносно високу швидкість.

Подібним чином бікубічна інтерполяція (BCI) [6] виконує кубічну інтерполяцію на кожній з двух вісей, як показано на рис. 1.6. Порівняно з BLI, BCI приймає 4 × 4 пікселі, що дає більш плавні результати з меншою кількістю артефактів, але набагато меншою швидкістю. Насправді BCI із згладжуванням є основним методом побудови наборів даних SR (тобто погіршення HR-зображень до LR-зображень) та також широко використовується в попередньому підвищенні дискретизації середовища SR. Власне кажучи, підвищення дискретизації на основі інтерполяційних методів покращують роздільну здатність зображення лише на основі його сигналів з власного зображення, не надаючи більше інформації.

Натомість вони часто вводять деякі побічні ефекти, такі як обчислювальна складність, посилення шуму, розмиття результатів. Тому сучасна тенденція полягає у заміні методів, заснованих на інтерполяції, шарами підвищення вибірки, що піддаються навчанню.

1.7.2 Підвищення дискретизації на основі навчання

Для того, щоб подолати недоліки методів, заснованих на інтерполяції, і вивчити наскрізне підвищення дискретизації, транспонований шар згортки та шар пікселів вводяться в поле SR.

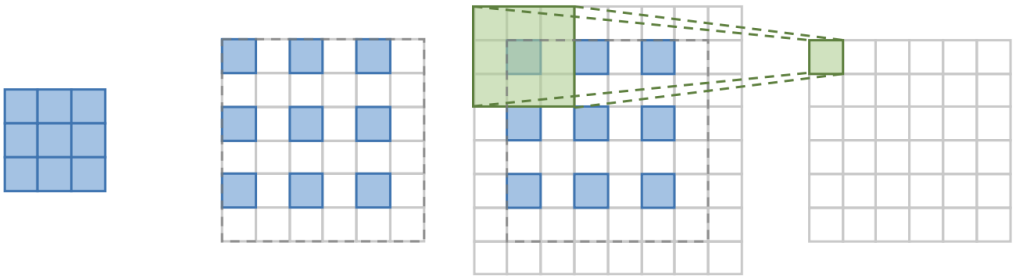


Рис. 1.7. Транспонований згортковий шар. Сині поля позначають початковий вид, а зелені поля вказують на ядро та результат згортки [1]

Транспонований шар згортки, намагається виконати трансформацію на противагу нормальній згортці, тобто прогнозування можливого введення даних на основі розмірів карт функцій, як вихід згортки. Зокрема, це збільшує роздільну здатність зображення шляхом розширення зображення, вставляючи нулі та виконуючи згортку. Беручи $2 \times SR$ з ядром 3×3 як приклад (рис. 1.7), початковий розмір спочатку розширюється вдвічі від оригінального розміру, де для доданих значень пікселів встановлено значення 0. Потім виконується згортка з ядром розміром 3×3, з кроком 1 і заміщенням 1. Таким чином, вхідні дані збільшено в 2 рази, у цьому випадку поле становить не більше 2×2. Оскільки транспонована згортка збільшує розмір зображення наскрізним способом, зберігаючи модель зв’язку, сумісна з звичайною згорткою, вона широко використовується як шари підвищення дискретизації в моделях надвисокої роздільної здатності. Однак цей шар може легко спричинити нерівномірність перекриття на кожній осі, і помножені результати на обох осях додатково створюють шаховий шаблон різної величини і тим самим знижує характеристики SR.

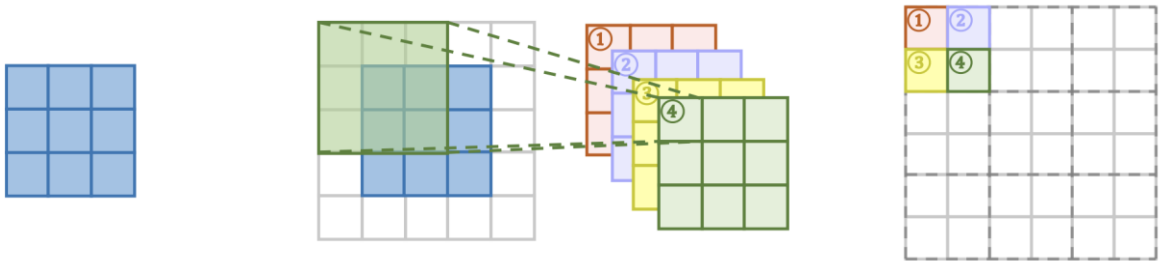


Рис 1.8. Шар Суб-піксельного збільшення дискретизації. Сині поля позначають вхідні дані, а поля з іншими кольорами позначають різні операції згортки і різний результуючий вид особливості карт [1]

Суб-піксельний шар, ще один наскрізний шар підвищення дискретизації, який можна навчити, виконує підвищення дискретизації шляхом генерації безлічі каналів за допомогою згортки і потім змінює їх форму, як показано на рис. 1.8. У межах цього шару а згортка спочатку застосовується для отримання результатів в s^2 каналів, де s - коефіцієнт масштабування. Якщо припустити, що розмір вводу $h \times w \times c$, вихідний розмір буде $h \times w \times s^2c$. Після цього виконується операція переформатування для отримання вихідних даних розміром $sh \times sw \times c$. У цьому випадку результуюче поле може мати розмір до 3×3. Завдяки наскрізному методу підвищення вибірки цей шар також є широко використовуваним моделями SR [13], [15].

Порівняно з транспонованим шаром згортки, шар пікселів має більш рецептивне поле, яке надає більше контекстної інформації, що допомагає створити більш реалістичні деталі. Однак оскільки розподіл рецептивних полів нерівномірний і блокові регіони насправді мають одне і те ж рецептивне поле, це може призвести до деяких артефактів поблизу кордонів різних блоків. З іншого боку, незалежно передбачення сусідніх пікселів у блоковій області може спричинити неоптимальні результати.

Таким чином шари підвищення дискретизації, засновані на навчанні, стали найбільш широко застосовуваними. Особливо в методі пост-дискретизації, ці шари зазвичай використовуються на завершальній фазі підвищення вибірки для реконструкції HR-зображень на основі уявлень високого рівня та витягуються в низько вимірному просторі, і таким чином досягаються кінцеві SR, уникаючи важких операцій у високо вимірному просторі.

Висновки до розділу 1

Перший розділ присвячений аналізу існуючих видів обробки зображень та дослідження методів отримання надвисокої роздільної здатності зображення які будуються на глибокому навчанні, що дозволить визначити необхідні сучасні технології для вирішення поставлених завдань. Зокрема:

1. Проведено дослідження загальної ієрархії структури методу для отримання надвисокої роздільної здатності зображення та математичний підхід для визначення даної структури.
2. Було виконано дослідження існуючих наборів даних для глибокого навчання нейромереж та визначено їх переваги та недоліки.
3. Розглянуто методи оцінки якості зображення, принцип їх визначення, побудови, характеристики, принципи роботи та їх переваги та недоліки
4. Досліджено параметри та характеристики зображення, та методи для оперування ними.
5. Поставлено та досліджено існуючі проблеми та недоліки при проектуванні методів для отримання надвисокої роздільної здатності зображень
6. Розглянуто та досліджено архітектури високого рівня для дискретизації зображень, визначено їх види, переваги та недоліки
7. Досліджено методи та види підвищення дискретизації зображення, принципи їх побудови переваги та недоліки.

Спираючись на проведене дослідження, аналіз методів та архітектурних рішень для збільшення роздільної здатності зображення, необхідно вирішити наступні завдання:

1. Визначити високорівневу архітектуру для побудови загального процесу обробки зображення та спроектувати метод на її основі
2. Розглянути та дослідити нейронні мережі, що вирішують задачі методів оцінки якості зображень та методів дискретизації на основі глибокого навчання.
3. Опираючись на переваги та недоліки використати обрані нейронні мережі для розробки рішення досліджених методів.

4. Спроектувати систему з використанням набору даних, що оптимально підходить для вирішення поставлених задач.
5. Визначити та розробити власну систему для отримання надвисокої роздільної здатності, що буде оптимально вирішувати поставлені задачі.
6. Оцінити ефективність розробленого методу, дослідити результати виконання та визначити переваги та недоліки, а також вказати на сфери та задачі, для використання в яких, цей метод підходить.

РОЗДІЛ 2
ПРОЕКТУВАННЯ АРХІТЕКТУРИ МЕТОДУ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ ЗОБРАЖЕННЯ

Багато сучасних моделей з надвисокою роздільною здатністю вивчають більшість функцій відображення низької роздільної здатності (LR) з наступним одним або кількома рівнями дискретизації в кінці мережі. Це називається Super Resolution (SR) після підвищення дискретизації на рис. 2.1. Хоча архітектура існуючих моделей дуже різниться, ми розглянемо дві найпоширеніші каркасні моделі на основі операцій підвищення дискретизації та їх розташування в моделі. Навчальні вибіркові шари доступні для навчання разом із попередніми згортковими шарами від початку до кінця.

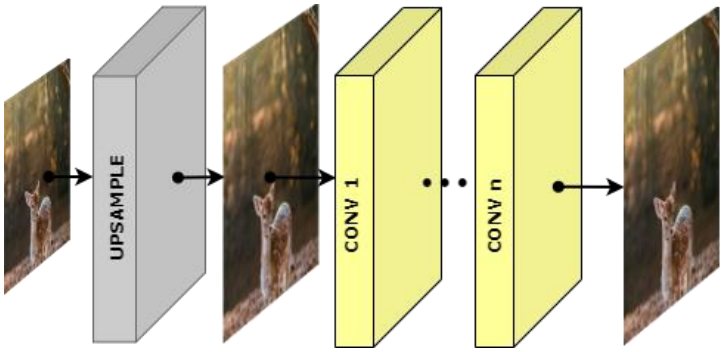


Рис. 2.1. Схема попереднього підвищення дискретизації для отримання надвисокої роздільної здатності

Попередня вибірка SR є складною, оскільки вона вивчає картографію з малого простору в зарозумілий простір, використовує традиційні алгоритми підвищення вибірки для отримання зображень з більш високою роздільною здатністю, а потім покращує їх за допомогою глибоких нейронних мереж. Зокрема, зображення LR збільшуються до зображення грубої високої роздільної здатності (HR) потрібного розміру, використовуючи звичайні методи (наприклад, бікубічна інтерполяція). Оскільки найскладніша операція підвищення дискретизації завершена, згортковій нейронній мережі (CNN) потрібно лише вдосконалити грубі зображення, що значно зменшує криву навчання.

Крім того, ці моделі можуть робити інтерпольовані зображення з довільними розмірами та коефіцієнтами масштабування як вхідні дані та забезпечують покращені результати із порівнянною продуктивністю з однорівневими моделями SR. Однак заздалегідь визначена підвищена дискретизація часто спричиняє побічні ефекти (наприклад, посилення шуму та розмиття), і оскільки більшість операцій виконуються в багатовимірному просторі, вартість часу до простору набагато вища, ніж інші рамки.

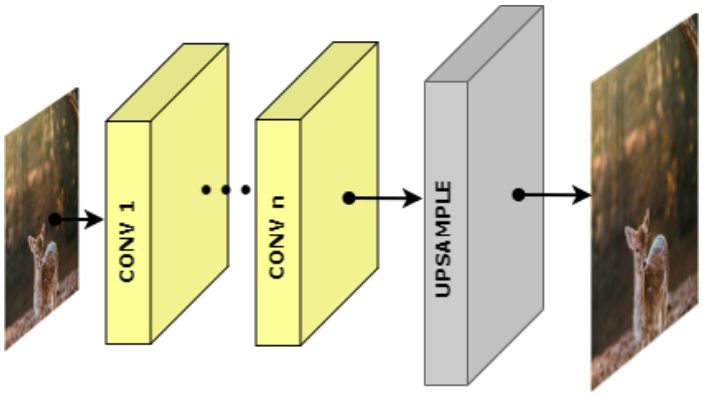


Рис. 2.2. Схема підвищення дискретизації після підвищення вибірки для отримання надвисокої роздільної здатності

Для підвищення обчислювальної ефективності та повного використання технології глибокого навчання для автоматичного збільшення роздільної здатності дослідники пропонують проводити більшу частину обчислень у низькорозмірному просторі, замінюючи заздалегідь визначене підвищення вибірки наскрізними навчальними рівнями, інтегрованими в кінці моделей. У SR, після підвищення дискретизації, як показано на рис. 2.1, видно, що вхідні LR-зображення подаються в глибокі CNN без збільшення роздільної здатності, а наскрізні навчальні шари підвищення дискретизації додаються в кінці мережі. Оскільки процес вилучення ознак з величезними обчислювальними витратами відбувається лише в низькорозмірному просторі, а роздільна здатність зростає лише в кінці, складність обчислень та просторів значно зменшується.

Раніше моделі спочатку проводили вибірку зображення LR, використовуючи заздалегідь визначену операцію підвищення вибірки, а потім досліджували відображення в HR-просторі (попереднє підвищення дискретизації для SR). Недоліком цього підходу є те, що для кожного шару

потрібно більше параметрів, що призводить до вищих обчислювальних витрат і обмежує побудову більш глибоких нейронних мереж.

2.1. Модель залишкової нейронної мережі

На сьогоднішній день побудова структури мережі була однією із найбільш важливих частин глибокого навчання. У галузі надвисокої роздільної здатності дослідники застосовують всі види стратегій проектування мережі поверх чотирьох SR архітектур (розділ 1.6) для побудови кінцевих мереж

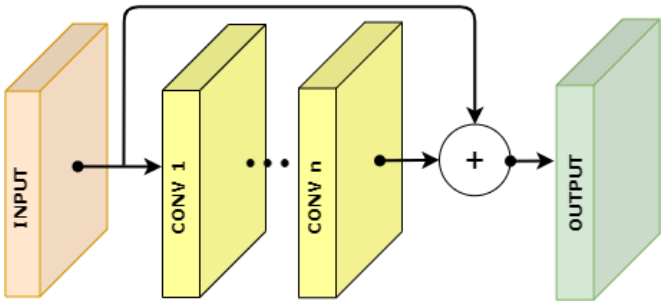


Рис. 2.3. Модель залишкової нейронної мережі

До того, як був запропонований ResNet для вивчення залишків замість ретельного картографування було проведено залишкове навчання широко застосовуваними моделями SR, як на рис. 2.3. Серед них можуть бути залишкові стратегії навчання, які поділяються на глобальне та локальне залишкове навчання.

Оскільки зображення SR це завдання переводу з зображення в зображення, де вхідне зображення дослідники намагаються високо корелювати цільові залишки між ними та отримати глобальне залишкове навчання. У цьому випадку даний процес дозволяє уникнути ускладненого перетворення з цілісного образу на інший, натомість потрібно лише вивчити залишкову карту для відновлення відсутніх високочастотних деталей. Оскільки залишки у більшості секторів наближаються до нуля, модель, складність і труднощі навчання значно зменшуються. Таким чином, він широко використовується для SR моделей.

Місцеве залишкове навчання подібне до залишкового навчання в ResNet [40] і використовується для полегшення проблеми деградації, спричиненої постійним збільшенням глибини мережі, зменшенням труднощів у навчанні та

покращеною здатністю до навчання. Він також широко використовується для SR.

На практиці обидва вищезазначені методи реалізовані за допомогою ярликових з'єднань (часто масштабується за допомогою невеликої константи) і стихійного додавання, тоді як різниця полягає в тому, що перший спосіб безпосередньо з'єднує вхідні та вихідні зображення, тоді як останній зазвичай додає кілька ярликів між шарами з різною глибиною всередині мережі.

На рис. 2.3. показано підключення до глобальної смуги пропускання на декількох рівнях. Ці рівні часто є залишковими блоками, як у Залишковій нейронній мережі (ResNet), або у спеціалізованих варіантах:

- Розширені глибокі залишкові мережі для надвисокої роздільної здатності одного зображення (EDSR)
- Широка активація для ефективної та точної роздільної здатності зображення (WDSR)

З’єднання локальної смуги пропускання в залишкових блоках спрощують оптимізацію мережі і, отже, підтримують побудову більш глибоких мереж.

2.2 Шар підвищення дискретизації

Для даної роботи шар, який ми будемо використовувати, це субпіксельний згортковий шар. Коли ви вводите розмір $H \times W \times C$ і коефіцієнт підвищення вибірки s, рівень згортання субпікселя спочатку створює подання розміру $H \times W \times Cs^2$ за допомогою операції згортки, а потім перетворює його до $sH \times sW \times C$, завершивши операцію підвищення дискретизації. Результат - просторово масштабований результат у s.

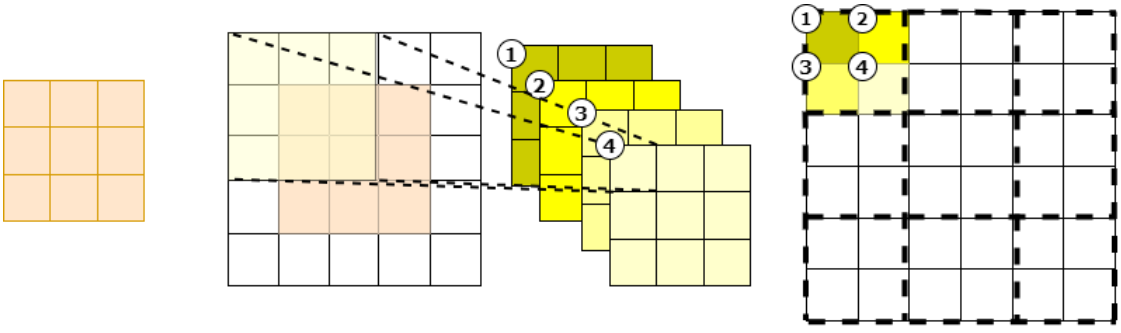


Рис 2.4. Послідовність субпіксельного збільшення дискретизації

Транспоновані згортки також можна вивчати та використовувати як альтернативу, але вони мають той недолік, що вони мають менше сприймальне поле, ніж субпіксельні згортки, і тому можуть обробляти менше контекстної інформації, що часто призводить до менш точних прогнозів.

2.3. Розширені глибокі залишкові мережі для надвисокої роздільної здатності одного зображення

Для вирішення проблеми надвисокої роздільної здатності ранні підходи використовують методи інтерполяції, засновані на теорії вибірки . Однак ці методи мають обмеження у прогнозуванні детальних, реалістичних текстур. Попередні дослідження застосували до проблеми статистику природних зображень для реконструкції кращих зображень із високою роздільною здатністю. Розширені роботи спрямовані на вивчення функцій відображення між парами зображень I^{LR} та I^{HR}. Ці методи навчання покладаються на прийоми, починаючи від вбудовування сусідів і до розрідженого кодування [41].

Деякі підходи використовують самоподібність зображень, щоб уникнути використання зовнішніх баз даних та збільшити розмір обмеженого внутрішнього словника шляхом геометричного перетворення латок. Нещодавно потужна здатність глибоких нейронних мереж призвела до значних поліпшень в SR [42]. Підключення через пропускну здатність та рекурсивна згортка полегшують тягар передачі інформації про особу в мережі надвисокої роздільної здатності. У багатьох алгоритмах глибокого навчання на основі роздільної здатності вхідне зображення збільшується за допомогою бікубічної інтерполяції до того, як воно подається в мережу. Замість використання інтерпольованого зображення в якості вхідних даних можливе також навчання модулям підвищення вибірки в самому кінці мережі. Роблячи це, можна зменшити більшу частину обчислень, не втрачаючи потужності моделі, оскільки розмір функцій зменшується. Однак у таких видів підходів є один недолік: вони не можуть вирішувати багатомасштабну проблему в єдиних рамках, як у VDSR [42]. У цій роботі ми вирішуємо дилему багатомасштабного навчання та обчислювальної ефективності. В EDSR використовується взаємозв'язок вивченої функції для

кожного масштабу, та нова багатомасштабна модель, яка ефективно реконструює зображення з високою роздільною здатністю для різних масштабів.

Середня квадратична помилка (MSE) або втрата L2 є найбільш широко використовуваною функцією втрат для загального відновлення зображення, а також є основним показником продуктивності (PSNR) для цих проблем. Однак тренування з втратою L2 не гарантує кращої продуктивності порівняно з іншими функціями втрат з точки зору PSNR та SSIM. У своїх експериментах мережа, навчена L1, досягла покращеної продуктивності порівняно з мережею, навченою L2.

2.3.1. Залишкові блоки у EDSR

Розширені глибинні залишкові мережі для надвисокої роздільної здатності одного зображення було обрано як модель, яка відповідає нашій архітектурі (рис. 2.4), оскільки вона має загальну архітектуру CNN із кількістю шарів B та кількістю F канали функцій, які займають приблизно $O(BF)$ пам'яті з параметрами $O(BF^2)$. Отже, збільшення F замість B може максимізувати продуктивність моделі, враховуючи обмежені обчислювальні ресурси. Залишкове масштабування застосовується до залишкового шляху, перш ніж додавати його назад до згорткового шляху. У кожному залишковому блоці після останнього згорткового шару розміщуються стійкі шари масштабування. На етапі тестування цей шар може бути інтегрований у попередній згортковий шар для підвищення обчислювальної ефективності. Немає шарів активації ReLU поза залишковими блоками, як показано на рис. 2.5 (b).

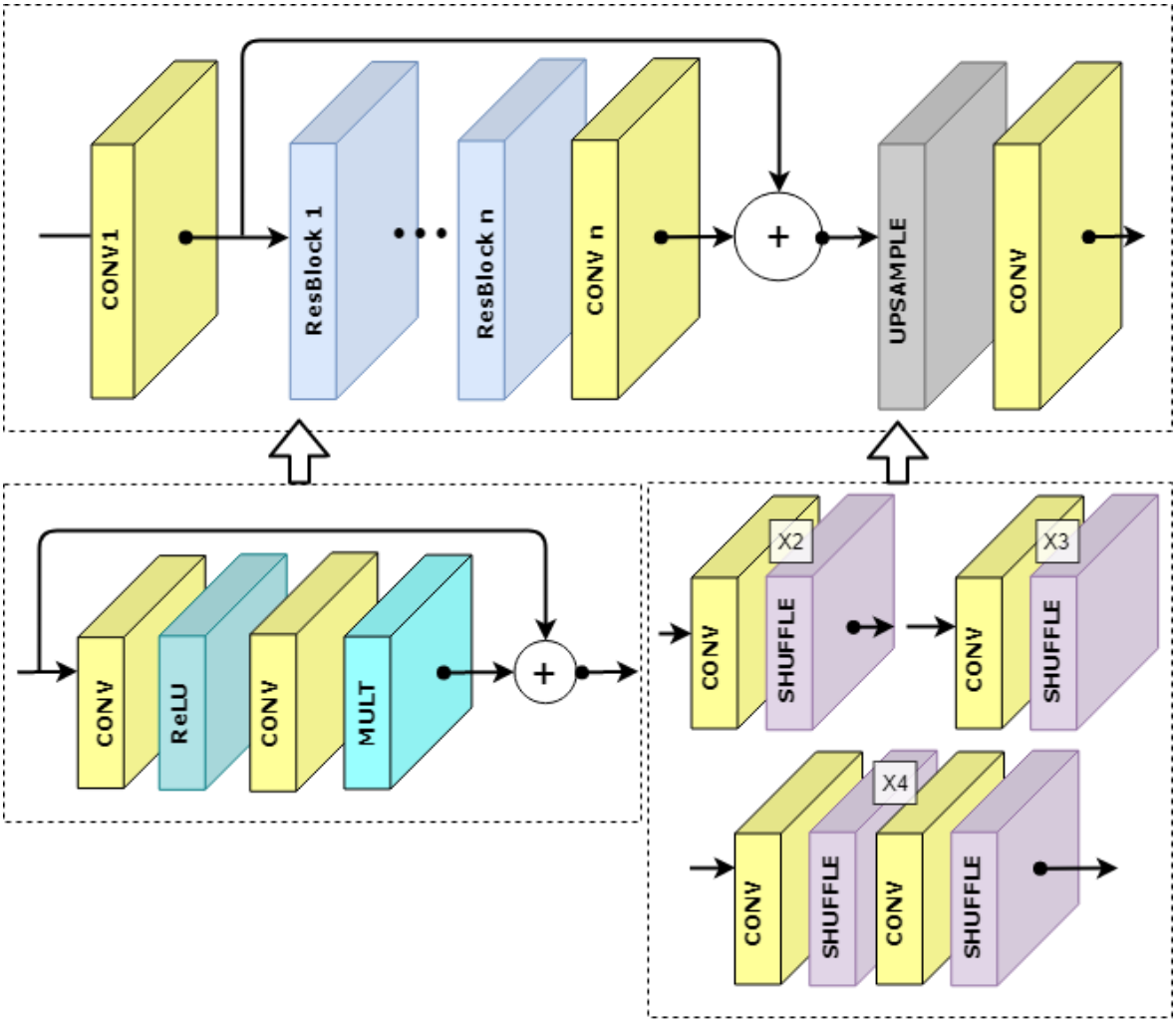


Рис. 2.5. Архітектура одномасштабної мережі SR (EDSR).

Пакетна нормалізація втрачає інформацію про масштаб зображення та зменшує гнучкість діапазону активацій, як описано авторами EDSR. Видалення шарів пакетної нормалізації не тільки збільшує продуктивність надвисокої роздільної здатності, але й зменшує пам’ять графічного процесора до 40%, що дозволяє навчати набагато більші моделі.

2.3.2 Одномасштабна модель EDSR

Найпростіший спосіб підвищити продуктивність моделі мережі - збільшити кількість параметрів. У згортковій нейронній мережі продуктивність моделі можна підвищити шляхом складання багатьох шарів або збільшенням кількості фільтрів. Однак збільшення кількості функціональних карт вище певного рівня зробить процедуру навчання чисельно нестабільною. У EDSR застосовують залишкове масштабування з коефіцієнтом 0,1. У кожному

залишковому блоці шари постійного масштабування розміщуються після останніх згорткових шарів. Ці модулі значно стабілізують навчальну процедуру при використанні великої кількості фільтрів.

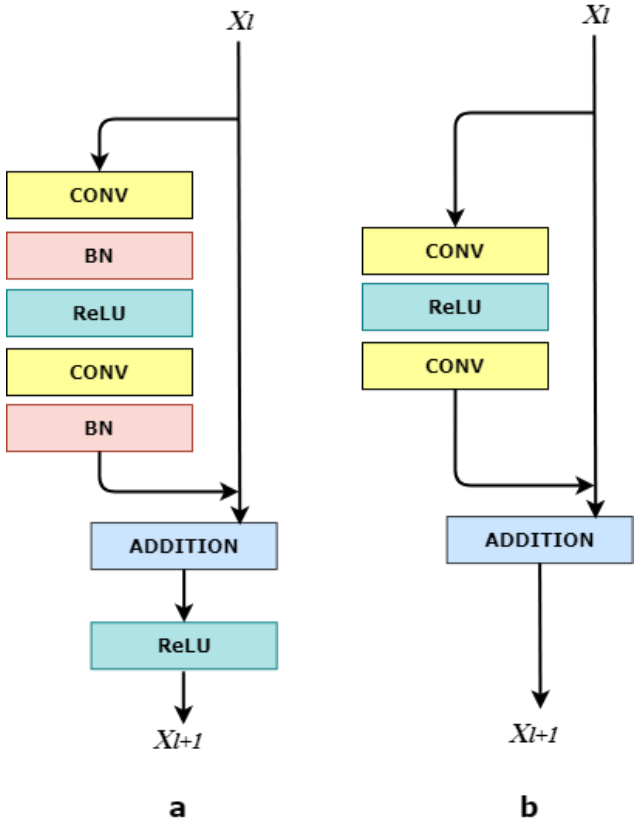


Рис. 2.6. Порівняння залишкових блоків, (a) ResNet, (b) EDSR

На етапі тестування цей рівень може бути інтегрований до попереднього рівня згортки для обчислювальної ефективності. Структура подібна до ResNet (рис 2.6), але модель EDSR не має шарів активації ReLU поза залишковими блоками. Крім того, у базовій моделі немає залишкових шарів масштабування, оскільки там використовується лише 64 карти функцій для кожного шару згортки.

У остаточній одномасштабній моделі (EDSR) базова модель була розширена, встановлюючи B = 32, F = 256 з коефіцієнтом масштабування 0,1. Архітектура моделі представлена на рис. 2.6. Під час підготовки моделі до коефіцієнта збільшення вибірки × 3 та × 4 ініціалізуються параметри моделі за допомогою попередньо навченої мережі × 2. Ця стратегія попередньої підготовки прискорює навчання та покращує остаточну ефективність, як це наочно продемонстровано на рис. 2.7. Для збільшення масштабу × 4, якщо

використовувати попередньо навчену модель масштабу × 2 (синя лінія), навчання сходиться набагато швидше, ніж починається з випадкової ініціалізації (зелена лінія).

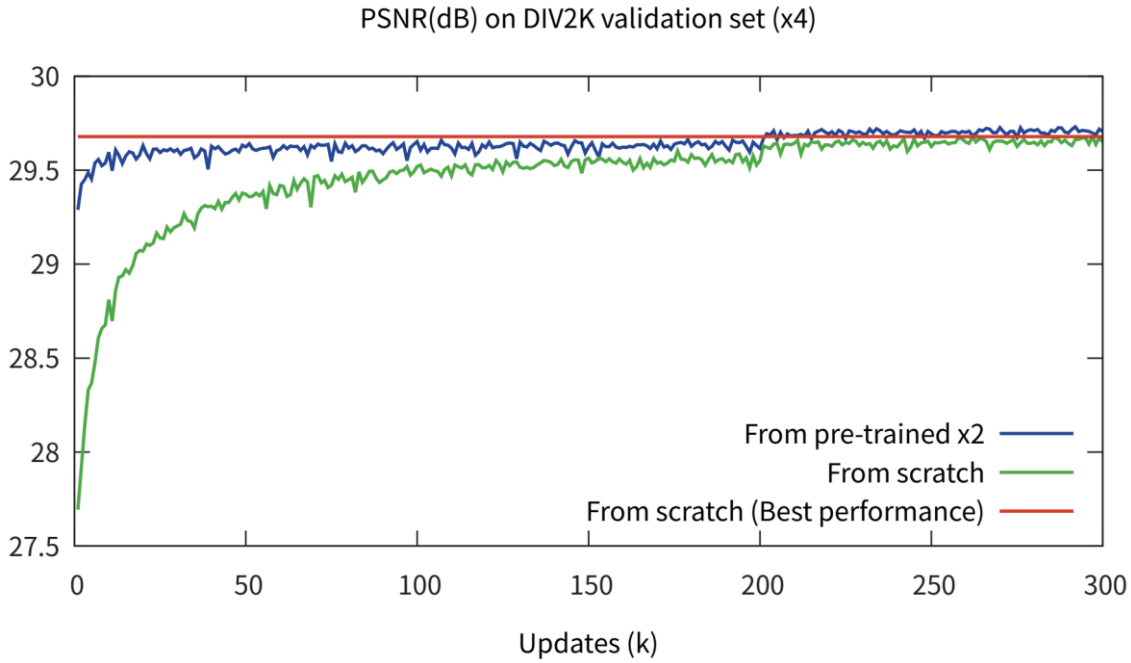


Рис. 2.7. Ефект від використання попередньо навченої мережі × 2 для × 4 моделі EDSR. Червона лінія вказує на найкращі показники зеленої лінії. Під час перевірки використовується 10 зображень для навчання [41]

Далі буде доцільно розглянути результати роботи даної нейронної мережі, та довести цим доцільність вибору її для нашої системи.

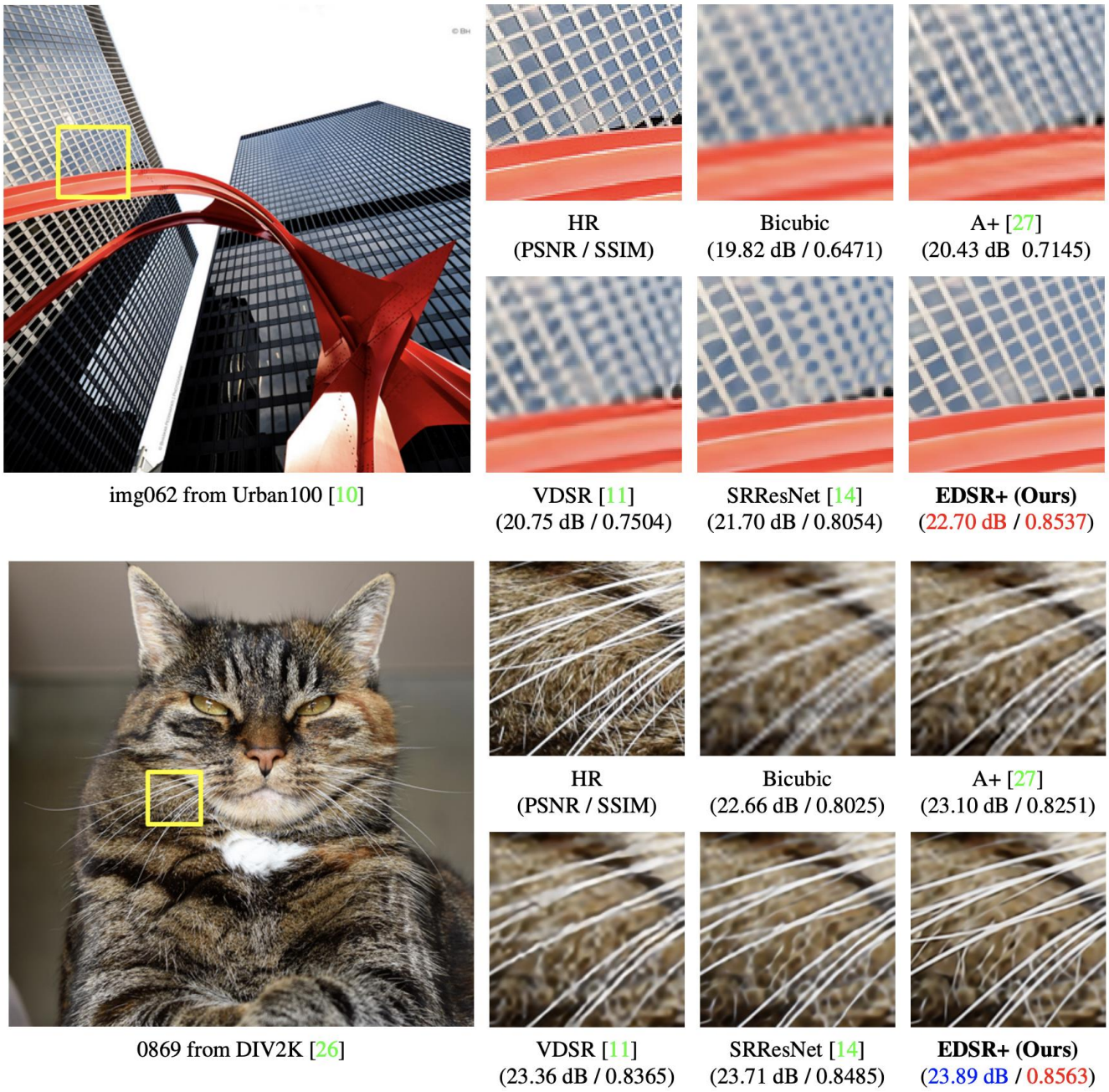


Рис. 2.8. Якісне порівняння моделі EDSR з іншими роботами для надвисокої роздільної здатності × 4 [41]

2.4. Широка активація для ефективної та точної роздільної здатності зображення

Глибокі згорткові нейронні мережі (CNN) були успішно застосовані до завдання надвисокої роздільної здатності одного зображення (SISR [44]). Метод SISR спрямований на відновлення зображення з високою роздільною здатністю (HR) із його аналога з низькою роздільною здатністю (LR) (як правило, бікубічна версія HR з зменшеною вибіркою). Він має багато застосувань у безпеці,

спостереженні, супутнику, медичній візуалізації і може служити вбудованим модулем для інших завдань відновлення або розпізнавання зображень. Попередні моделі для отримання зображення з надвисокою роздільною здатністю, включаючи SRCNN [45], FSRCNN [46], ESPCN [47], використовували відносно неглибокі згорткові нейронні мережі (з глибиною від 3 до 5).

Вони поступаються за точністю порівняно з запропонованими пізніше глибокими мережами SR (наприклад, ResNet [48] та EDSR [49]). Збільшення глибини приносить свої певні переваги, але тим часом недостатньо використовується інформація про об'єкти з неглибоких шарів (як правило, це низькорівневі функції). Щоб вирішити цю проблему, методи, включаючи SRDenseNet [50], RDN [51], MemNet [52], вводять різні пропуски з’єднань та операцій конкатенації між неглибокими шарами та глибокими шарами, формалізуючи цілісні структури для надвисокої роздільної здатності зображення. Замість того, щоб додавати різні сполучення ярликів, нелінійні ReLU перешкоджають потоку інформації від дрібних шарів до глибших.

На основі залишкової мережі SR, без додаткових параметрів та обчислень, просто розширення можливостей перед активацією ReLU призводить до значних поліпшень для надвисокої роздільної здатності одного зображення, перемагаючи мережі SR зі складними пропусками та конкатенаціями, включаючи SRDenseNet та MemNet. Інтуїція в виконанні роботи полягає в тому, що розширення можливостей до ReLU дозволяє пропускати більше інформації, зберігаючи при цьому вкрай нелінійність глибоких нейронних мереж. Таким чином, низькорівневі SR-функції з неглибоких шарів можуть бути простішими для поширення до кінцевого шару для кращих прогнозів щільного значення пікселів.

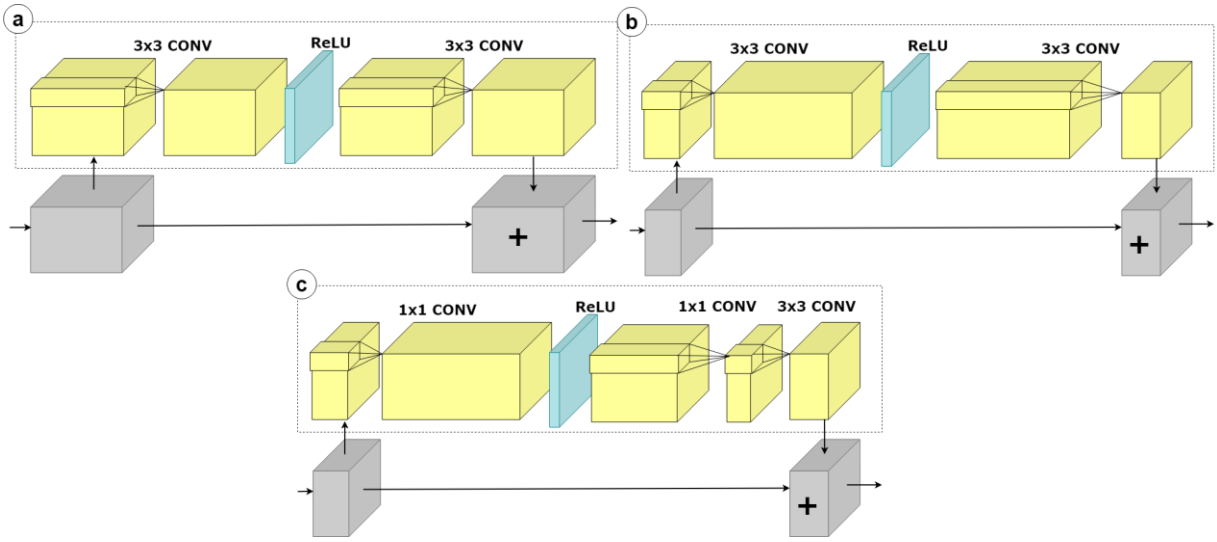


Рис. 2.9. (a): звичайний залишковий блок, що використовується в EDSR. (b) WDSR-A, (c) WDSR-B

На рис. 2.9 продемонстровано WDSR-A: залишковий блок із широкою активацією, який має тонкий шлях відображення ідентичності з більш широкими (від ×2 до ×4) каналами перед активацією в кожному залишковому блоці. WDSR-B: залишковий блок з більш широкою активацією (від ×6 до ×9) без обчислювальних витрат та лінійною згорткою низького рангу. У WDSR A та WDSR-B усі рівні активації ReLU застосовуються лише між двома широкими функціями (функції з більшим номером каналів).

Основна ідея широкої активації змушує досліджувати ефективні способи розширення функцій до ReLU, оскільки просто додавання більшої кількості параметрів неефективне для сценаріїв SR зображення в режимі реального часу [43]. SR-залишкова мережа WDSR A, має тонкий шлях відображення ідентичності з більш широкими (від ×2 до ×4) каналами перед активацією в кожному залишковому блоці. Однак коли коефіцієнт розширення перевищує 4, канали шляху відображення ідентичності повинні бути ще зменшені, це різко погіршує точність. Таким чином, як другий крок, зберігається постійна кількість каналів шляху відображення ідентичності. Отже розробники WDSR пропонують лінійну згортку низького рангу, яка розкладає велике ядро згортки на два ядра згортки низького рангу. Завдяки більш ширшій активації та лінійним згортками низького рангу, будується мережа SR WDSR-B. Вона має ширшу активацію (від ×6 до ×9) без додаткових параметрів або обчислень, і додатково підвищує

точність для надвисокої роздільної здатності зображення. Ілюстрація WDSR A та WDSR-B наведена на рисунку 2.9.

Експерименти показують, що ширша активація послідовно перевершує їх базові лінії при різних бюджетах параметрів. Крім того, у порівнянні з пакетною нормалізацією, або відсутністю нормалізації, навчання з нормалізацією ваги призводить до кращої точності для мереж із глибокою надвисокою роздільною здатністю. Попередні роботи, включаючи EDSR, виявили, що нормалізація пакетів погіршує точність SR, що також підтверджено в експериментах.

Однак із збільшенням глибини нейронних мереж для SR (наприклад, MDSR має глибину близько 180), мережі без нормалізації партії стають важкими для навчанн. Нормалізація ваги дозволяє тренувати мережу SR на порядок вищу швидкість навчання, що призводить як до швидшої конвергенції, так і до кращої продуктивності.

2.4.1. Широка активація для WDSR-A мережі

У даному розділі ми описуємо розширення функції до рівня активації ReLU без обчислювальних накладних витрат та ефекти широкої активації всередині залишкового блоку для WDSR-A мережі. Простий спосіб - це безпосереднє додавання номерів каналів усіх функцій. Однак це нічого не доводить, крім того, що більша кількість параметрів призводить до кращої продуктивності.

Таким чином ми обираємо мережу SR для вивчення важливості широких функцій перед активацією з однаковими параметрами та обчислювальними бюджетами. Наш перший крок до широкої активації надзвичайно простий: ми зменшуємо особливості шляху відображення залишкової ідентичності, одночасно розширюючи функції до активації, як показано на рис. 2.9. Двошарові залишкові блоки спеціально вивчаються за базовим EDSR. Припустимо, що ширина шляху відображення ідентичності (рис. 2) дорівнює w_1, а ширина до активації всередині залишкового блоку дорівнює w_2. Ми вводимо коефіцієнт розширення перед активацією як r, таким чином $w_2 = r \times w_1$. У ванільних залишкових мережах (наприклад, використовуваних в EDSR та MDSR) ми маємо

$w_2 = w_1$, а кількість параметрів становить $2 \times w_1^2 \times k^2$ у кожному залишковому блоці. Складність обчислень (операції додавання) - це постійне масштабування номерів параметрів, коли ми фіксуємо розмір вхідного патчу. Ця проста ідея формує широко активовану SR-мережу WDSR-A. Експерименти показують, що WDSR-A надзвичайно ефективна для підвищення точності SISR, коли r становить від 2 до 4. Однак для r, що перевищує цей поріг, продуктивність швидко падає. Це, ймовірно, через занадто тонкий шлях відображення ідентичності. Наприклад, у нашому базовому EDSR (16 залишкових блоків з 64 фільтрами) для надвисокої роздільної здатності ×3, коли r перевищує 6, w_1 буде навіть менше, ніж кінцевий простір подання HR-зображення $S^2 * 3$ (використовується зміна порядку пікселів як шар вибірок)) де S - коефіцієнт масштабування, а 3 - RGB. Таким чином, ми отримуємо ефективне згортання параметрів для подальшого підвищення точності та ефективності завдяки ширшому активуванню.

2.4.2. Ефективна Широка активація для WDSR-B мережі

Щоб вирішити вищезазначене обмеження, підтримуються постійні номери каналів шляху відображення ідентичності. Зокрема, 4 згортки 1×1 широко використовуються для розширення чи зменшення номера каналу в ResNets [53], ResNeXts [54] та MobileNetV2 [55]. У WDSR-B (рис. 2.9) спочатку розширюються номери каналів, використовуючи 1×1, а потім застосовується нелінійність (ReLU) після шару згортки. Далі виконується ефективна лінійна згортка низького рангу, яка розкладає велике ядро згортки на два ядра згортки низького рангу. Це стек з однією згорткою 1×1 для зменшення кількості каналів та однією згорткою 3×3 для здійснення просторового вилучення функцій. Додавання активації ReLU в лінійних згортках низького рангу значно знижує точність, що також підтримує широку гіпотезу активації.

2.4.3 Структура мережі WDSR

Глобальний залишковий шлях являє собою лінійний стек із декількох згорткових шарів, що використовує багато обчислювальних ресурсів. Лінійні

згортки є зайвими (рис. 2.10) і можуть певною мірою поглинатися в залишкове тіло.

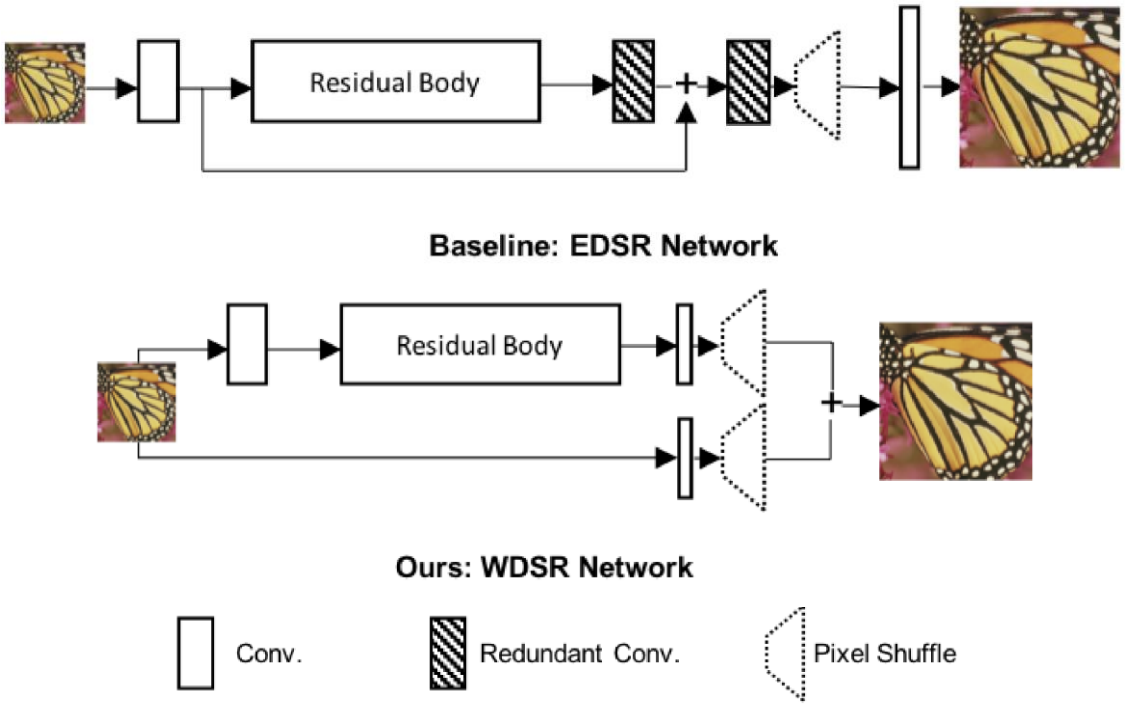


Рис. 2.10. Демонстрація спрощеної мережі WDSR у порівнянні з EDSR [43]

Таким чином у мережевій структурі використовується одиночний згортковий шар із розміром ядра 5×5, який безпосередньо приймає $3\times H\times W$ LR RGB зображення як вхід і вихід $3S^2\times H\times W$ HR аналоги, де S - масштаб . Це призводить до зменшення параметрів та обчислень. У експериментах не було виявлено жодного зниження точності з простішою формою. Шар збільшення дискретизації на відміну від попереднього сучасного рівня техніки, де один або кілька згорткових шарів вставляються після повторної дискретизації, WDSR витягує всі функції на стадії низької роздільної здатності (рис. 2.10). Емпірично було виявлено, що це не впливає на точність мереж SR, а значно покращує швидкість.

2.4.4 Шари нормалізації

Для демонстрації ефективності нормалізації ваги для вдосконаленого навчання SR-мереж. Ми порівнюємо точність навчання та тестування (PSNR), коли тренуємо одну і ту ж модель з різними методами нормалізації, тобто

нормалізацією ваги, нормалізацією партії або відсутністю нормалізації. Результати на рис. 2.11 показують, що модель, що навчається з нормалізацією ваги, має швидшу конвергенцію та кращу точність. Модель, навчена з нормалізацією партії, нестабільна під час тестування, що, ймовірно, зумовлено різними рецептурами BN у процесі навчання та тестування.

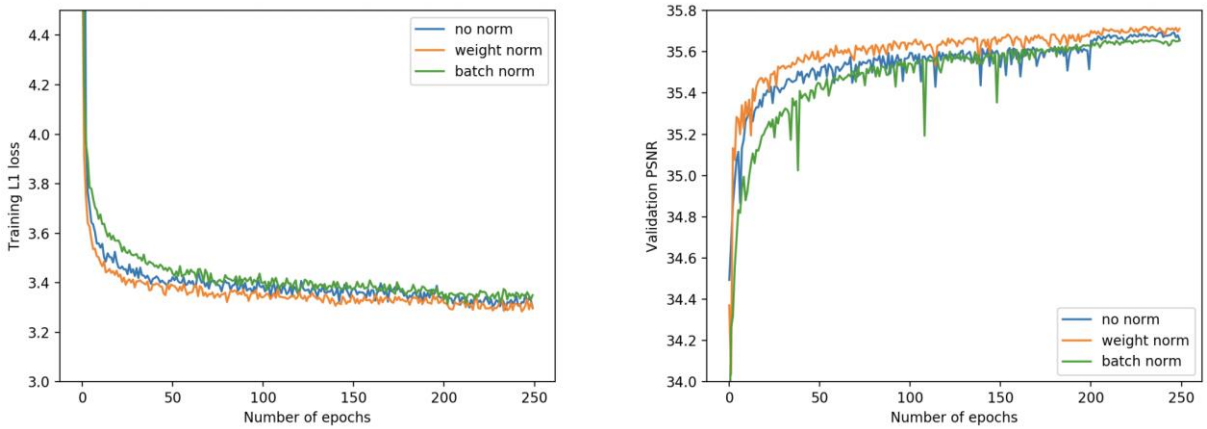


Рис. 2.11. Тренування *L1* втрати та перевірки PSNR тієї ж моделі, яка тренується з нормалізацією ваги, нормалізацією партії або без нормалізації [43]

Для подальшого навчання, оскільки швидкість занадто велика для моделей, які навчаються із пакетною нормалізацією, також навчається одна і та ж модель з різними показниками навчання. Результати показані на малюнку 2.12. Навіть при $lr = 10^{-4}$, коли криві тренувань стабільні, перевірка PSNR все ще не є стабільною протягом тренувань.

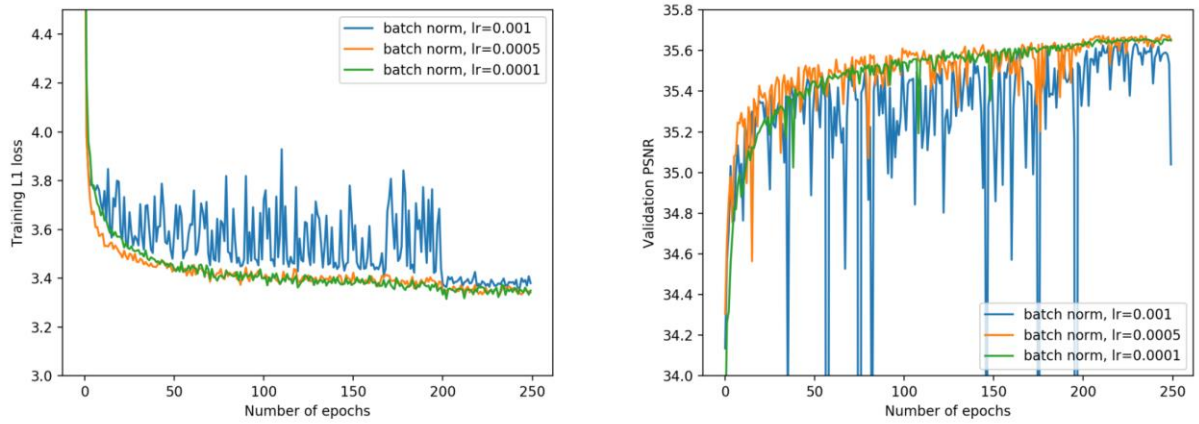


Рис. 2.11. Навчання *L1* втрати та перевірки PSNR моделі, навченої з нормалізацією партії, але різною швидкістю навчання [43]

Висновки до розділу 2

Отже в рамках цього розділу ми визначили та спроектували та дослідили базову архітектуру для методу підвищення якості зображення, для її розробки були обрані наступні структурні елементи:

1. В якості схеми для отримання SR зображення була обрана структура з пост-підвищенням дискретизації, оскільки вона має відносно більш точний результат та забезпечує стабільну вибірку
2. Для шару підвищення дискретизації було обрано метод субпіксельної дискретизації, так як він може обробляти більше контекстної інформації
3. Модель залишкової нейронної мережі була обрана оскільки має простоту та прозорість отримання результату.
4. В якості основної мережі для отримання зображення надвисокої роздільної здатності була обрана EDSR мережа, оскільки вона побудована на методі згорткових мереж та має один з найкращих якісних результатів.
5. WDSR виступає як активаційний шар, та має масу переваг над аналогами, та був розроблений, як покращення активаційного шару у порівнянні з цим елементом у EDSR.

Спираючись на виконане проектування потрібно зосередити увагу на наступних завданнях:

- Використовуючи спроектовану архітектуру, покращити її в ході розробки.
- Дослідити та використати сучасні нейромережі для використання спроектованої архітектури у задачі покращення якості відеозображення.
- Дослідити отримані результати розробленого методу та оцінити доцільність його використання у поставленій задачі.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СПРОЕКТОВАНОЇ АРХІТЕКТУРИ МЕТОДУ ДЛЯ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕННЯ

3.1. Вимоги до програмного та апаратного забезпечення

Для реалізації описаного вище методу нам необхідно:

1. Апаратне забезпечення, що має достатню для навчання нейромереж продуктивність.
2. Програмне забезпечення для створення необхідного середовища, де можна запускати допоміжні засоби для подальшої роботи.
3. Програмне забезпечення, як середовище для запуску програмного коду, що дозволяє пройти всі кроки навчання для нейромереж.

Для виконання даного завдання, в якості апаратного забезпечення було вирішено використовувати локальний ПК з наступними характеристиками:

- 8-ми ядерний процессор Intel i7 9700k з тактовою частотою 4,9 ГГц
- 32ГБ RAM, DDR4, 3600 МГц, CL17
- Графічний процесор Nvidia RTX 2080 AMP
- Операційна система Windows 10

В якості програмного забезпечення, оскільки такі аналоги як хмарні обчислення мають недостатню кількість часу сесії для повного проходження всіх кроків навчання, наприклад GoogleCollab, в якості середовища для запуску та виконання програмного коду, було вирішено використовувати Anaconda, як дистрибутив для виконання програмного коду на мові програмування Python та JupyterNotebook відповідно.

3.1.1. Anaconda

Anaconda - це дистрибутив мов програмування Python і R для наукових обчислень (наукові дані, програми машинного навчання, широкомасштабна обробка даних, прогнозована аналітика тощо), що спрямована на спрощення управління та розгортання пакетів. Дистрибутив включає пакети з обробки даних, придатні для Windows, Linux та macOS. Він розробляється та підтримується Anaconda, Inc., яка була заснована Пітером Вангом і Тревісом

Оліфантом у 2012 році. Як продукт Anaconda, Inc., він також відомий як Anaconda Distribution або Anaconda Individual Edition, тоді як іншими продуктами компанії є Anaconda Team Edition та Anaconda Enterprise Edition, обидва з яких не є безкоштовними версіями пакетів в Anaconda керує система управління пакетами conda. Цей менеджер пакетів був виділений як окремий пакет із відкритим кодом, оскільки в підсумку він був корисним як для себе, так і для інших речей. Існує також невелика, завантажувальна версія Anaconda під назвою Miniconda, яка включає лише conda, Python, пакети, від яких вони залежать, і невелику кількість інших пакетів.

Дистрибутив Anaconda постачається з автоматично встановленими понад 250 пакетами, а з PyPI можна встановити понад 7500 додаткових пакетів з відкритим кодом, а також пакетом conda та менеджером віртуального середовища. Він також включає графічний інтерфейс користувача, Anaconda Navigator, як графічну альтернативу інтерфейсу командного рядка (CLI).

Велика різниця між conda та менеджером пакетів pip полягає в тому, як управляються залежностями пакетів, що є значним викликом для науки даних Python та причиною існування conda [56].

До версії 20.3, коли pip встановлював пакет, він автоматично встановлював будь-які залежні пакунки Python, не перевіряючи, чи конфліктують вони з раніше встановленими пакетами. Він встановив би пакет та будь-які його залежності, незалежно від стану існуючої інсталяції. Через це користувач із робочою інсталяцією, наприклад, Google Tensorflow, міг виявити, що він перестав працювати, використовуючи pip для встановлення іншого пакета, для якого потрібна інша версія залежною бібліотеки numpy, ніж та, що використовується Tensorflow. У деяких випадках пакет, здається, працює, але детально дає різні результати. Хоча з тих пір pip реалізував послідовне вирішення залежностей, ця різниця пояснює історичну диференціацію менеджера пакетів conda.

На відміну від цього, conda аналізує поточне середовище, включаючи все встановлене на даний момент, і разом із зазначеними обмеженнями версії (наприклад, користувач може побажати мати Tensorflow версії 2,0 або новішої),

розробляє, як встановити сумісний набір залежностей, і відображає попередження, якщо цього зробити не вдається.

Пакети з відкритим кодом можна встановити окремо із сховища Anaconda, Anaconda Cloud (anaconda.org) або власного приватного сховища або дзеркала користувача, використовуючи команду conda install. Anaconda, Inc. компілює та створює пакети, доступні у самому сховищі Anaconda, та забезпечує двійкові файли для Windows 32/64 bit, Linux 64 bit та MacOS 64-bit. Все, що є на PyPI, може бути встановлене в конда-середовище за допомогою pip, і conda буде відслідковувати те, що він сам встановив і що pip встановив.

Спеціальні пакети можуть бути зроблені за допомогою команди conda build, і ними можна поділитися з іншими, завантаживши їх у Anaconda Cloud, [16] PyPI або інші сховища.

За замовчуванням установка Anaconda2 включає Python 2.7, а Anaconda3 включає Python 3.7. Однак можна створити нові середовища, які включають будь-яку версію Python, упаковану з conda.

Anaconda Navigator - графічний користувальницький інтерфейс (GUI) для робочого столу, що входить до дистрибутиву Anaconda, що дозволяє користувачам запускати програми та керувати пакетами conda, середовищами та каналами без використання команд командного рядка. Навігатор може шукати пакунки на Anaconda Cloud або в локальному сховищі Anaconda, встановлювати їх у середовищі, запускати пакети та оновлювати. Він доступний для Windows, macOS та Linux.

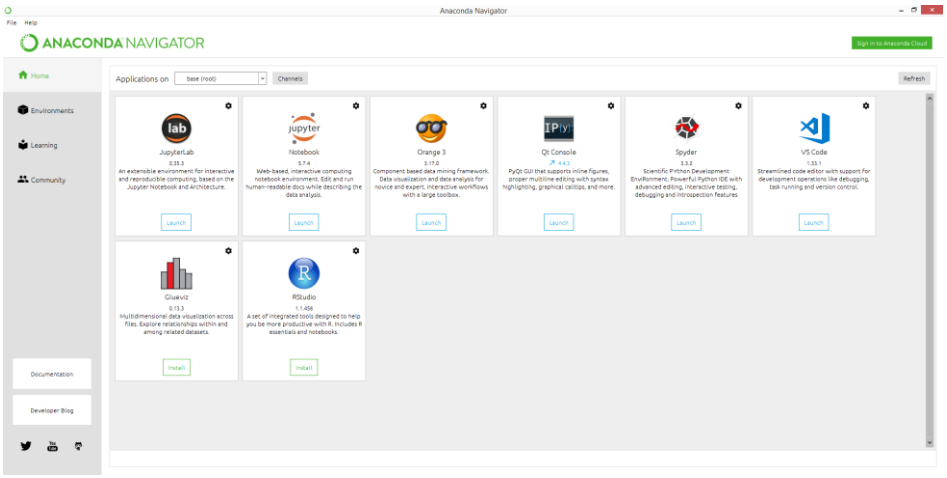


Рис. 3.1. Користувацький інтерфейс Anaconda Navigator

Наступні програми доступні за замовчуванням у Навігаторі:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code

3.1.2. Jupyter Notebook

Jupyter Notebook (раніше IPython Notebooks) - це веб-інтерактивне обчислювальне середовище для створення документів для ноутбуків Jupyter. Термін «Notebook» («блокнот») може в розмовній формі посилатися на багато різних сутностей, головним чином веб-додаток Jupyter, веб-сервер Jupyter Python або формат документа Jupyter залежно від контексту. Документ Jupyter Notebook - це документ JSON, який слідує за версійною схемою, що містить упорядкований список клітинок вводу / виводу, який може містити код, текст, математику, графіки та мультимедійні файли, як правило, закінчуючись розширенням ".ipynb" .

Блокнот Jupyter можна перетворити на безліч відкритих стандартних форматів виводу (HTML, слайди презентацій, LaTeX, PDF, ReStructuredText, Markdown, Python) за допомогою "Завантажити як" у веб-інтерфейсі через бібліотеку nbconvert або «jupyter nbconvert» інтерфейс командного рядка в оболонці. Щоб спростити візуалізацію документів ноутбуків Jupyter в веб-браузері, бібліотека nbconvert надається як послуга через NbViewer, яка може взяти URL-адресу до будь-якого загальнодоступного документа блокнота, перетворити його в HTML та відобразити в користувачеві.

Jupyter Notebook надає REPL на основі браузера, побудований на ряді популярних бібліотек з відкритим кодом:

- IPython
- ØMQ (ZeroMQ)
- Tornado (web server)
- jQuery
- Bootstrap (front-end framework)
- MathJax

Jupyter Notebook може підключатися до багатьох ядер, щоб дозволити програмування на різних мовах. За замовчуванням Jupyter Notebook постачається з ядром IPython. Станом на 2.3 версію було 49 сумісних з Jupyter ядер для багатьох мов програмування, включаючи Python, R, Julia та Haskell [57].

Jupyter Notebook подібний до обчислювального стилю інтерфейсу ноутбука інших програм, таких як Maple, Mathematica та SageMath.

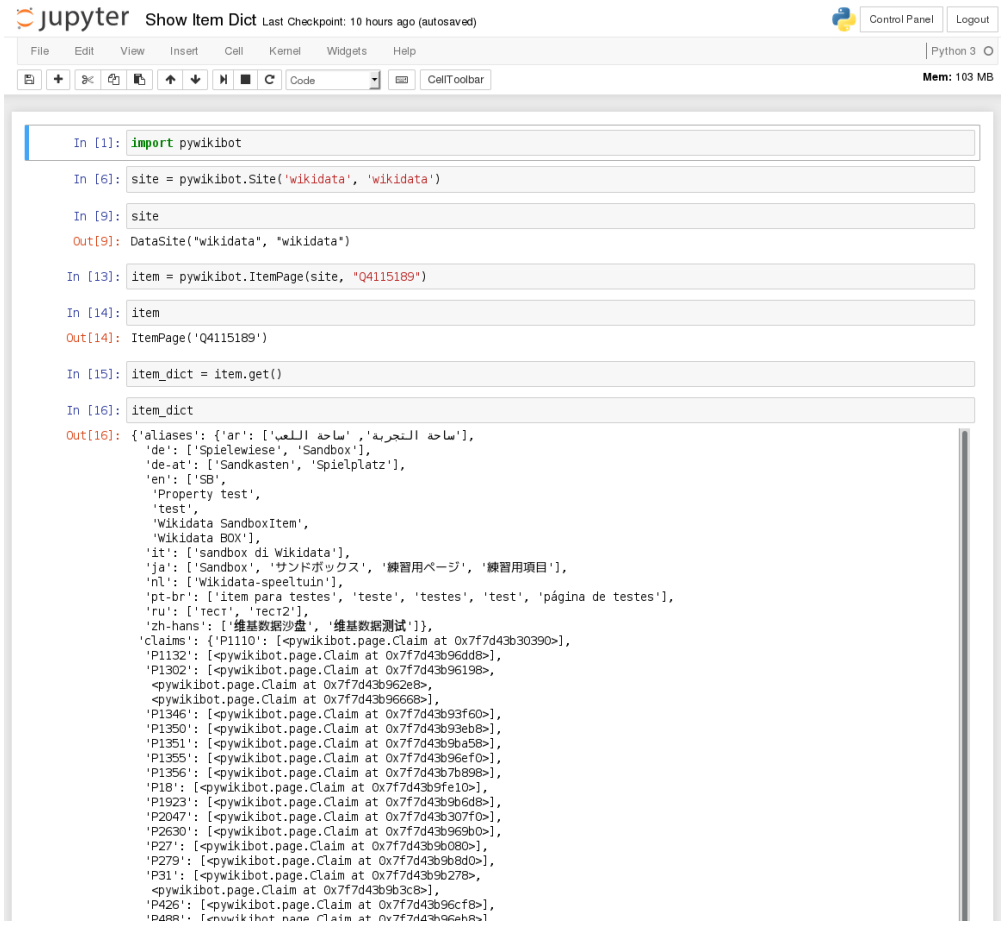


Рис. 3.2. Користувацький інтерфейс JupyterNotebook

Таким чином ми визначили необхідні для реалізації технології, та обгрунтовано обрали їх для подальшої розробки. В якості бібліотеки, яка включає в себе усі раніше описані технології машинного навчання, ми обрали Tensorflow.

3.2. Tensorflow

TensorFlow - це безкоштовна бібліотека програмного забезпечення з відкритим кодом для машинного навчання. Вона може бути використана для цілого ряду завдань, але приділяє особливу увагу навчанню та висновку про глибокі нейронні мережі. Символічна математична бібліотека, заснована на потоці даних та диференційованому програмуванні. Вона використовується як для досліджень, так і для виробництва в Google.

TensorFlow - це система Google Brain другого покоління. Версія 1.0.0 була випущена 11 лютого 2017 р. Хоча еталонна реалізація працює на окремих пристроях, TensorFlow може працювати на декількох процесорах та графічних процесорах (з додатковими розширеннями CUDA та SYCL для обчислень загального призначення на графічних процесорах). TensorFlow доступний на 64-розрядних платформах Linux, macOS, Windows та мобільних обчислювальних платформах, включаючи Android та iOS.

Його гнучка архітектура дозволяє легко розгортати обчислення на різних платформах (процесори, графічні процесори, TPU), а також від робочих столів до кластерів серверів до мобільних та крайніх пристроїв.

Обчислення TensorFlow виражаються у вигляді графіків потоку даних. Назва TensorFlow походить від операцій, які такі нейронні мережі виконують над багатовимірними масивами даних, які називаються тензорами [58].

3.3 Дані для навчання нейромереж

Приклад надвисокої роздільної здатності одного зображення (SR) спрямована на повне відновлення насичених деталей (високих частот) у зображеннях на основі попередніх прикладів у вигляді зображень із низькою роздільною здатністю (LR) та відповідних зображень із високою роздільною здатністю (HR). Втрата деталей / вмісту може бути наслідком різних факторів,

що погіршують стан, таких як розмиття, децимація, шум або апаратні обмеження (наприклад, датчики камери). Ефективність найкращих методів постійно покращувалась. Існує постійна потреба у стандартизованих контрольних показниках SR, що дозволяють порівняти різні запропоновані методи за однакових умов. Більшість нещодавніх SR-робіт прийняли кілька наборів даних, таких як 91 зображення поїздів [59].

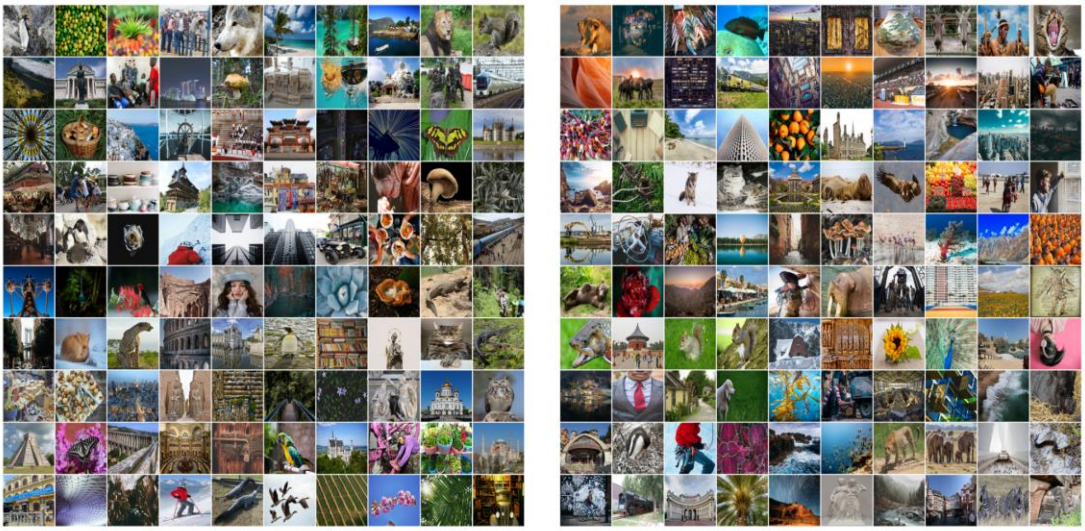


Рис. 3.3. Візуалізація запропонованих перевірочних та тестових зображень DIV2K. DIV2K містить також 800 зображень поїздів [59]

Набір даних DIV2K, новий набір зображень із роздільною здатністю DIVerse 2K для порівняльного тестування орієнтованого одержуваного зображення з надвисокою роздільною здатністю (рис. 3.4) та подальшого збільшення (вмісту) розробники вручну просканували 1000 кольорових RGB-зображень з Інтернету, приділяючи особливу увагу якості зображення, різноманітності джерел (сайтів та камер), вмісту зображень та авторським правам.

DIV2K призначений для дослідницьких цілей. Усі 1000 зображень мають роздільну здатність 2K, тобто вони мають 2K пікселів принаймні на одній з осей (вертикальної чи горизонтальної). Всі зображення оброблялися за допомогою тих самих інструментів. Для простоти, оскільки найпоширеніші фактори збільшення в SR має розміри × 2, × 3 та × 4, автори DIV2K обрізали зображення на множину 12 пікселів за обома осями. Більшість сканованих зображень

спочатку перевищували 20 млн пікселів. Зображення мають високу якість як в естетичному плані маленька кількість шуму та інші пошкодження (наприклад, розмиття та зміна кольору). DIV2K охоплює велику різноманітність вмісту, починаючи від людей, предметів та середовищ ручної роботи (міста, села), флори та фауни, а також природних пейзажів, включаючи умови під водою та недостатнє освітлення.

Після збору зображень DIV2K 1000 була обчислена ентропія зображення, біт на піксель (bpp), ступінь стиснення PNG та оцінки CORNIA та застосовано бікубічне зменшення масштабу × 3, а потім збільшення масштабу × 3 з бікубічною інтерполяцією (функція імрезації Matlab), Методи ANR та A+ та налаштування за замовчуванням. Автори випадковим чином генерували розділи з 800 поїздів, 100 перевірок та 100 тестових зображень, поки не досягли гарного балансу спочатку візуального вмісту, а потім середньої ентропії, середнього bpp, середньої кількості пікселів на зображення (ppi), середніх показників якості CORNIA, а також у відносних відмінностях між середніми показниками PSNR бікубічного, ANR та A+ методів. На рис. 3.4 узагальнено основні характеристики перевірки та тестових розділів DIV2K у порівнянні з найпопулярнішими наборами даних SR. Рис. 3.3 візуалізує 100 зображень для перевірки та 100 зображень для тестування набору даних DIV2K.

Dataset	images	CORNIA	ppi	bpp PNG	entropy
Set5	5	21.50 (±20.00)	113491	12.45 (±1.77)	7.37 (±0.51)
Set14	14	21.56 (±12.52)	230202	12.57 (±3.89)	7.03 (±0.89)
B100	100	18.46 (±12.74)	154401	14.12 (±2.76)	7.30 (±0.44)
Urban100	100	5.80 (±19.26)	774313	14.01 (±2.76)	7.57 (±0.36)
DIV2K validation	100	15.86 (±15.38)	2835028	12.69 (±2.69)	7.33 (±0.83)
DIV2K test	100	17.47 (±14.55)	2757182	12.64 (±2.33)	7.50 (±0.47)
DIV2K test ⇓ 2 bic.	100	17.93 (± 13.09)	689295	13.41 (± 2.38)	7.49 (± 0.47)
DIV2K test ⇓ 4 bic.	100	18.26 (± 13.08)	172323	14.40 (± 2.45)	7.48 (± 0.47)
DIV2K test ⇓ 8 bic.	100	23.20 (± 14.09)	43166	15.37 (± 2.44)	7.47 (± 0.46)
DIV2K test ⇓ 16 bic.	100	31.34 (± 15.76)	10849	16.49 (± 2.32)	7.45 (± 0.45)
DIV2K test ⇓ 2 crop	100	16.93 (± 14.55)	689295	13.79 (± 2.45)	7.52 (± 0.35)
DIV2K test ⇓ 4 crop	100	22.59 (± 15.75)	172323	14.09 (± 2.67)	7.37 (± 0.45)
DIV2K test ⇓ 8 crop	100	30.85 (± 18.66)	43166	14.12 (± 3.06)	7.14 (± 0.62)
DIV2K test ⇓ 16 crop	100	41.32 (± 19.76)	10849	14.20 (± 3.49)	6.88 (± 0.82)

Рис. 3.4. Основні характеристики наборів даних SR. Вказано середнє та стандартне відхилення [59]

3.4. VGG-19

Основною моделлю для архітектури та порівняння є дуже глибокі згорткові мережі для широкомасштабного розпізнавання зображень.

VGG була створена групою візуальної геометрії в Оксфорді. Вона несе і використовує деякі ідеї своїх попередників, вдосконалює їх і використовує глибокі згорткові нейронні шари для підвищення точності.

Перш за все давайте розберемо, що таке ImageNet. Це база даних Image, що складається з 14 197 122 зображень, організованих відповідно до ієрархії WordNet. це ініціатива для допомоги дослідникам, студентам та іншим у галузі дослідження зображень та зору. ImageNet також проводить конкурси, серед яких був масштабний виклик візуального розпізнавання ImageNet (w), який кинув виклик дослідникам по всьому світу запропонувати рішення, що дають найнижчі показники помилок топ-1 та топ-5 (коефіцієнт помилок в топ-5% зображень, де правильна мітка не є однією з п’яти найбільш вірогідних міток моделі). Конкурс дає 1000 навчальних комплектів з 1,2 мільйона зображень, набір для перевірки 50 тисяч зображень та тестовий набір із 150 тисяч зображень і ось виходить архітектура VGG [60].

Отже, простою мовою VGG - це глибокий CNN, який використовується для класифікації зображень. Шари в моделі VGG19 продемонстровані на рис. 3.5.

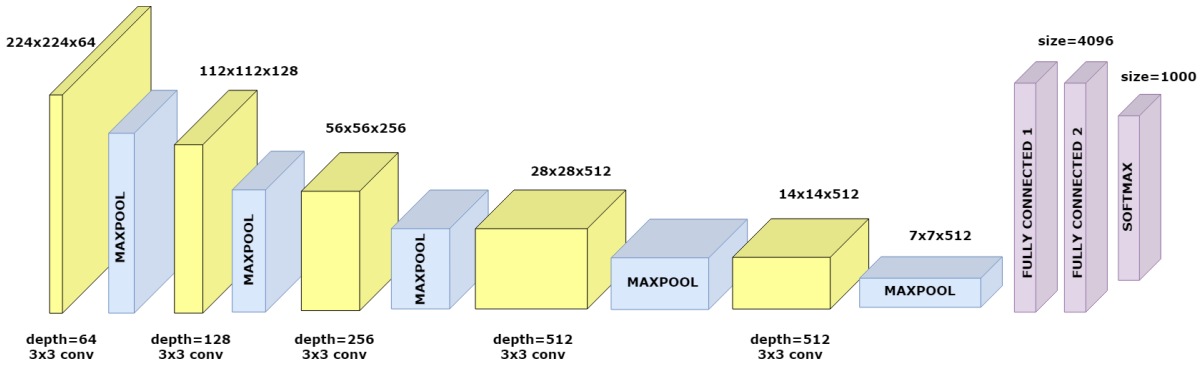


Рис. 3.5. Модель нейромережі VGG19

Фіксований розмір (*224 * 224*) зображення RGB був наданий як вхід для цієї мережі, що означає, що матриця мала форму (*224 224,3*). Єдиною попередньою обробкою, яку було зроблено, було те, що вони віднімали середнє значення RGB з кожного пікселя, обчислене протягом усього навчального набору.

Використані ядра розміром (*3 * 3*) із розміром кроку 1 піксель, це дозволило їм охопити все поняття зображення. Просторове заповнення використовувалось для збереження просторової роздільної здатності зображення. Об'єднання макс. було виконано у вікнах *2 * 2* пікселі з *sride 2*, за цим слідувала випрямлена лінійна одиниця (ReLu), щоб ввести нелінійність, щоб зробити модель класифікацією кращою, та покращити обчислювальний час, оскільки попередні моделі використовували функції тана або сигмоподібної області, це виявилося набагато кращим, ніж ті. Реалізовано три повністю з'єднаних шари, з яких перші два мали розмір 4096, а потім шар з 1000 каналами для 1000-смугової класифікації ILSVRC, а кінцевий шар є функцією softmax.

Понад 14 мільйонів зображень з бази даних ImageNet було використано для навчання мережі VGG-19. Ця попередньо навчена мережа може класифікувати до 1000 об’єктів. Мережа пройшла навчання з кольорових зображень розміром *224×224* пікселів. Але сьогодні він трохи застарів і використовує досить велику кількість ресурсів, наприклад, не підходить для використання на мобільних пристроях.

3.5. MobileNetV2

Модель MobileNet розроблена для використання в мобільних додатках, і це перша модель мобільного комп'ютерного зору TensorFlow. MobileNet використовує відокремлені поглиблення звивини. Це значно зменшує кількість параметрів у порівнянні з мережею зі звичайними звивинами з однаковою глибиною в сітках Це призводить до полегшення глибоких нейронних мереж. Згортка, що відокремлюється по глибині, складається з двох операцій.

- Глибока згортка.
- Точкова згортка.

MobileNet - це клас CNN, який був відкритий для Google, і тому це дає нам чудову відправну точку для навчання наших класифікаторів, які шалено малі та шалено швидкі. Швидкість та енергоспоживання мережі пропорційні кількості MAC (Multiply-Accumulates), що є мірою кількості сплавлених операцій множення та додавання.

Основний блок цієї мережі в цілому схожий на попереднє покоління, але має ряд ключових особливостей. Як і в MobileNetV1, він має згорткові блоки кроків 1 і 2 (рис. 3.6). Блоки з кроком 2 призначені для зменшення просторових розмірів тензора і, на відміну від блоку з кроком 1, не мають залишкових з'єднань [61].

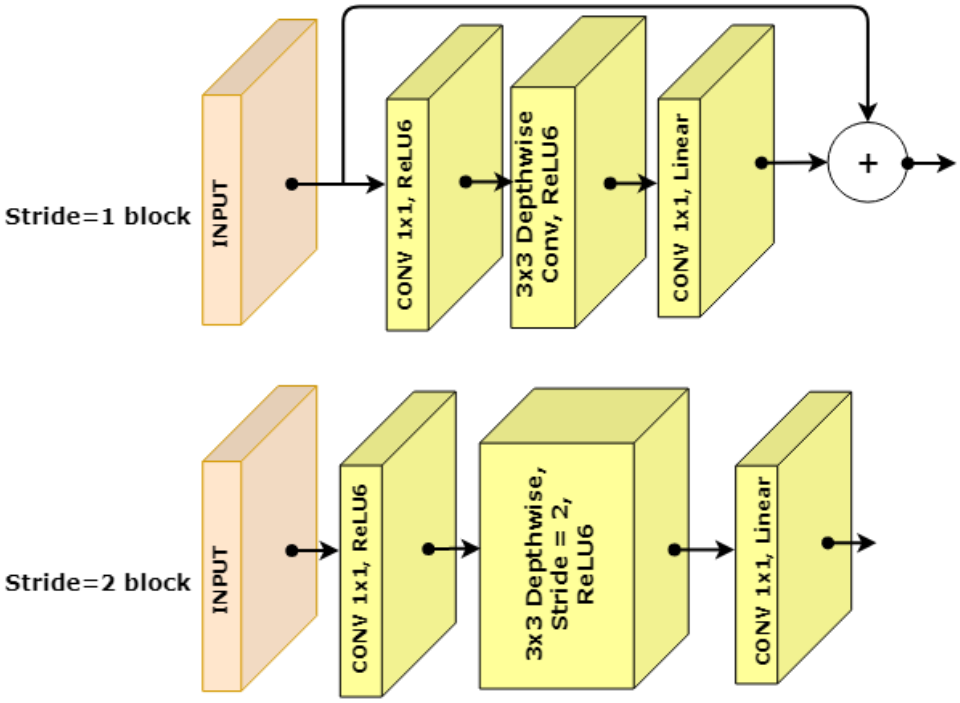


Рис. 3.6. Архітектура мережі MobileNet2

3.5.1 Глибоко відокремлена згортка

Ця згортка виникла з ідеї, що глибину та просторовий розмір фільтра можна розділити - таким чином, назва відокремлювана. Візьмемо приклад фільтру Собеля, який використовується при обробці зображень для виявлення країв.

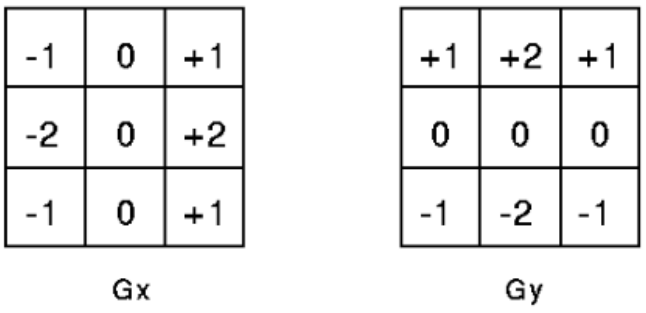


Рис. 3.6. Фільтр Собеля. *Gx* для вертикального краю, *Gy* для виявлення горизонтального краю [62]

Ми можемо розділити розміри висоти та ширини цих фільтрів. *Gx*-фільтр можна розглядати як матричний добуток [1 2 1] транспонування за допомогою [-1 0 1]. Ми помічаємо, що фільтр маскувався. Це показує, що він мав дев'ять параметрів, але має 6. Це стало можливим завдяки розділенню його розмірів висоти та ширини.

Та сама ідея, що застосовується для відокремлення виміру глибини від горизонталі (ширина * висота), дає нам відокремлену згортку, коли ми виконуємо згортку по глибині. Після цього ми використовуємо фільтр 1*1 для покриття розміру глибини. В цьому процесі слід звернути увагу на те, наскільки параметри зменшуються цією згорткою, щоб вивести те саме число. каналів. Щоб створити один канал, нам потрібні 3*3*3 параметра для виконання згортки по глибині та 1*3 параметра для подальшого вимірювання глибини в згортці.

Але якщо нам потрібні три вихідних канали, нам потрібен лише фільтр глибини 31 * 3, що дає нам загалом 36 (27+9) параметрів, тоді як для того самого немає. вихідних каналів у звичайній згортці нам потрібні фільтри 33 * 3 * 3, що дають нам загалом 81 параметр. Глибоко відокремлена згортка - це згортка по глибині, за якою йде точкова згортка, як показано на рис. 3.7.

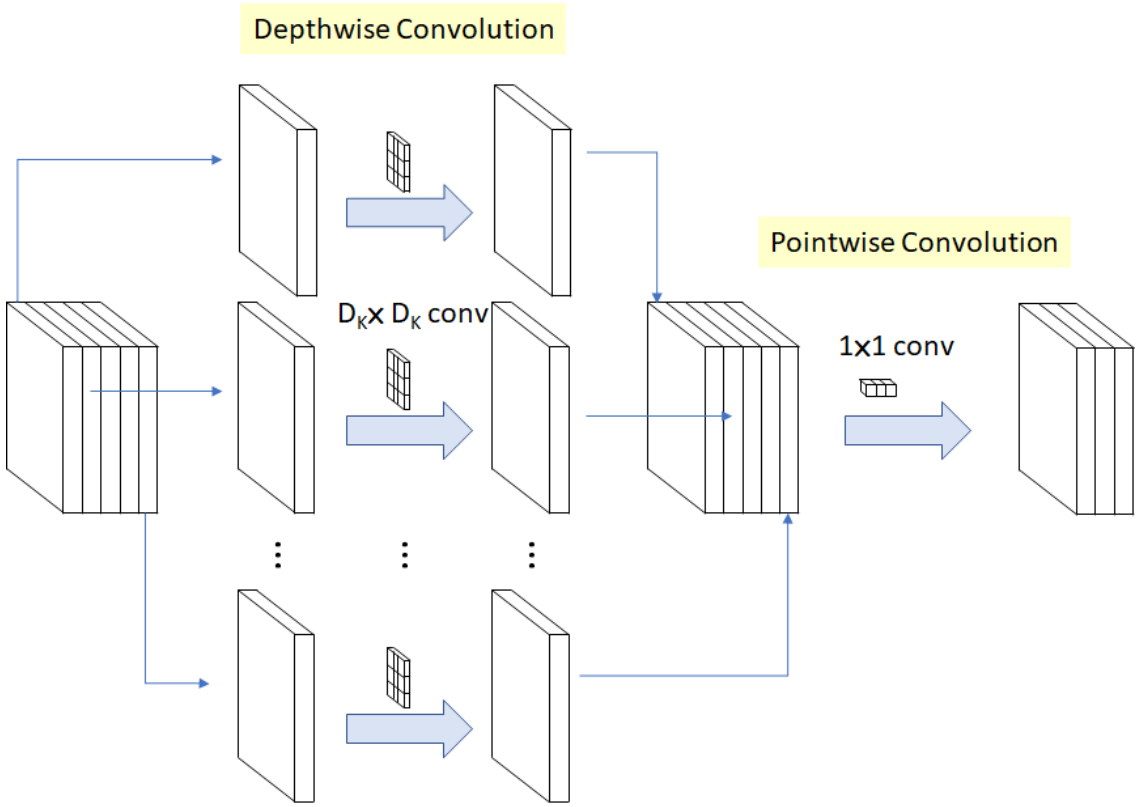


Рис. 3.7. Глибока відокремлена згортка (Depthwise Separable Convolution) [62]

Глибока згортка - це просторова згортка по каналу $DK \times DK$ (рис. 3.8). Нехай на малюнку вище ми маємо п’ять каналів; тоді ми матимемо 5 $DK \times DK$ просторових згорток. Також можна охарактеризувати, як карта єдиної згортки на кожному вхідному каналі окремо. Тому її кількість вихідних каналів така ж, як і кількість вхідних каналів. Його обчислювальна вартість становить $Df^2 * M * Dk^2$

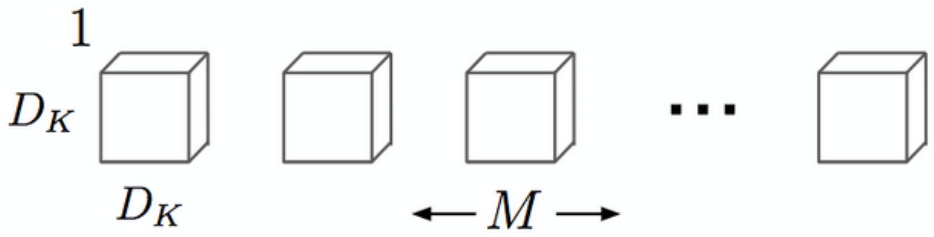


Рис. 3.8. Глибока згортка [62]

Точкова згортка - це згортка 1 × 1 для зміни розмірності (рис. 3.9). Згортка з розміром ядра 1x1, яка просто поєднує функції, створені глибокою згорткою. ЇЇ обчислювальна вартість становить $M * N * Df^2$.

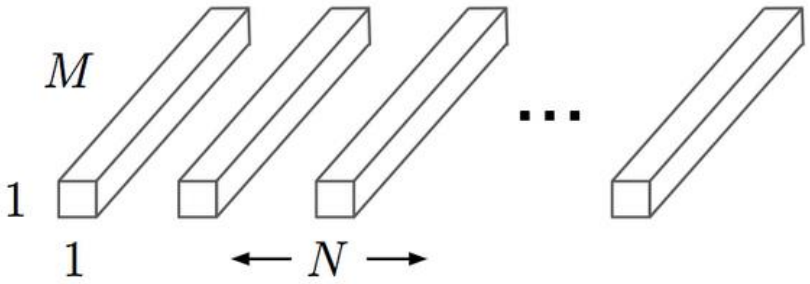


Рис. 3.9. Точкова згортка [63]

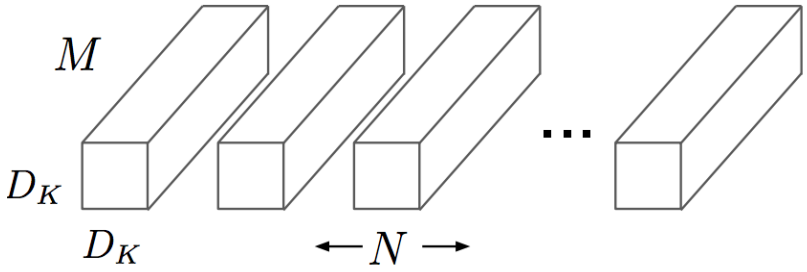


Рис. 3.10. Схема звичайної згортки [62]

Основна відмінність між архітектурою MobileNet та традиційною CNN (рис. 3.10) замість єдиного згорткового шару 3x3, за яким слід йдуть пакетна норма та ReLU. Мобільні мережі розділяють згортку на конвекцію по глибині 3x3 та конвекцію в точці 1x1, як показано на рис. 3.11.

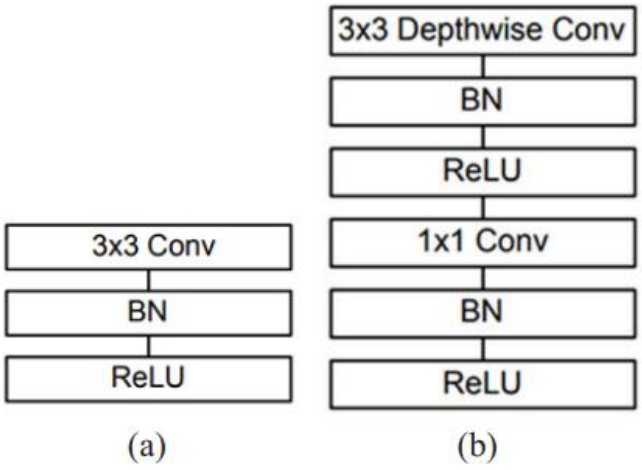


Рис. 3.11. (a) - Стандартний згортковий шар із нормалізацією партії та ReLU (b) - Розгортання згортки по глибині з глибокими та точковими шарами з наступною нормалізацією партії та ReLU [62]

Таким чином ми використовуємо для розробки та порівняння MobileNet, оскільки це сімейство перших мобільних моделей комп'ютерного зору для TensorFlow, розроблених для ефективної максимальної точності, пам’ятаючи про обмежені ресурси для вбудованого або вбудованого додатка. Також це невеликі моделі з низькою затримкою та малою потужністю, параметризовані для задоволення обмежень ресурсів у різних випадках використання. Вони можуть використовуватися для класифікації, виявлення, вбудовування та сегментації.

3.6. EfficientNet

Модель EfficientNet була запропонована командою з Google Research, у своїй дослідницькій роботі Rethinking Model Scaling for Convolutional Neural Networks. Цей документ був представлений на Міжнародній конференції з машинного навчання у 2019 році. Ці дослідники вивчили масштабування моделі та виявили, що ретельне збалансування глибини, ширини та роздільної здатності мережі може призвести до кращої продуктивності.

На основі цього спостереження вони запропонували новий метод масштабування, який рівномірно масштабує всі розміри глибини, ширини та роздільної здатності мережі. Вони використали пошук нейронної архітектури для проектування нової базової мережі та масштабували її, щоб отримати сімейство моделей глибокого навчання, які називаються EfficientNets, що забезпечують набагато кращу точність та ефективність у порівнянні з попередніми Світовими нейронними мережами.

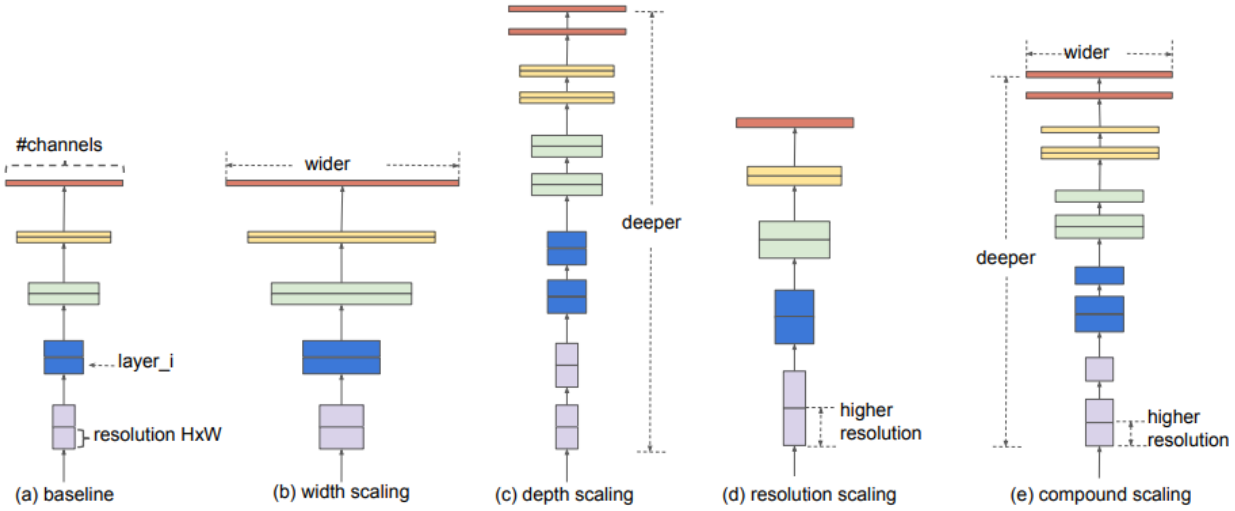


Рис. 3.12. Масштабування моделі. (a) - є прикладом базової мережі; (b) - (d) - це звичайне масштабування, яке збільшує лише один вимір мережі ширина, глибина або роздільна здатність. (e) - запропонований нами метод складання масштабу, який рівномірно масштабує всі три виміри з фіксованим співвідношенням [63]

У своїй роботі дослідники стверджували, що цей метод масштабування покращив точність та ефективність моделі. Спочатку дослідники спроектували базову мережу, виконавши пошук нейронної архітектури, техніку для автоматизації проектування нейронних мереж. Це оптимізує як точність, так і ефективність, виміряну на основі операцій з плаваючою точкою в секунду (FLOPS).

Існує 8 реалізацій EfficientNet, починаючи від B0 - B7, оскільки архітектура стає більш складною. Тим не менше, навіть найпростіший EfficientNetB0 працює добре. Маючи лише 5,3 мільйона параметрів, він забезпечує високу точність, тому налаштування моделі машинного навчання не займе багато часу. Отримана

архітектура використовує мобільну інвертовану вузьку згортку (MBConv), схожу на MobileNetV2 та MnasNet, але трохи більшу через збільшення бюджету (FLOPS). Потім вони масштабують базову мережу, щоб отримати сімейство моделей під назвою EfficientNets (рис. 3.13).

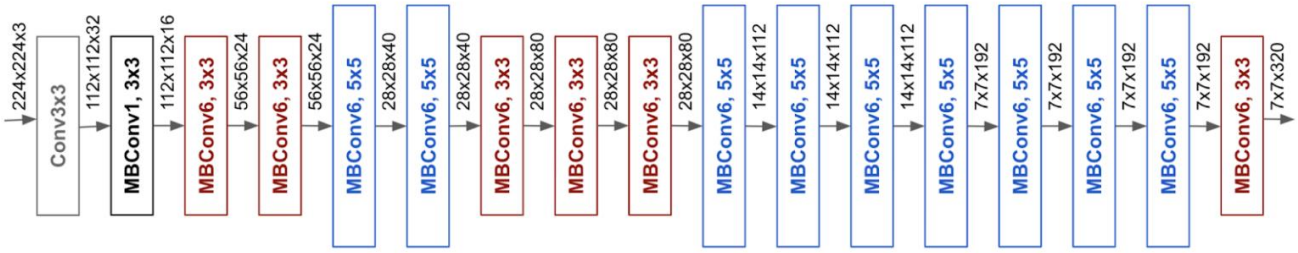


Рис. 3.13. The architecture for baseline network EfficientNet-B0 [63]

Вони також представили порівняння продуктивності EfficientNet з іншими потужними моделями навчання передачі, коли працювали над набором даних ImageNet. Було показано, що остання версія EfficientNet, тобто EfficientNet-B7, має найвищу точність серед усіх із меншою кількістю параметрів (рис. 3.14).

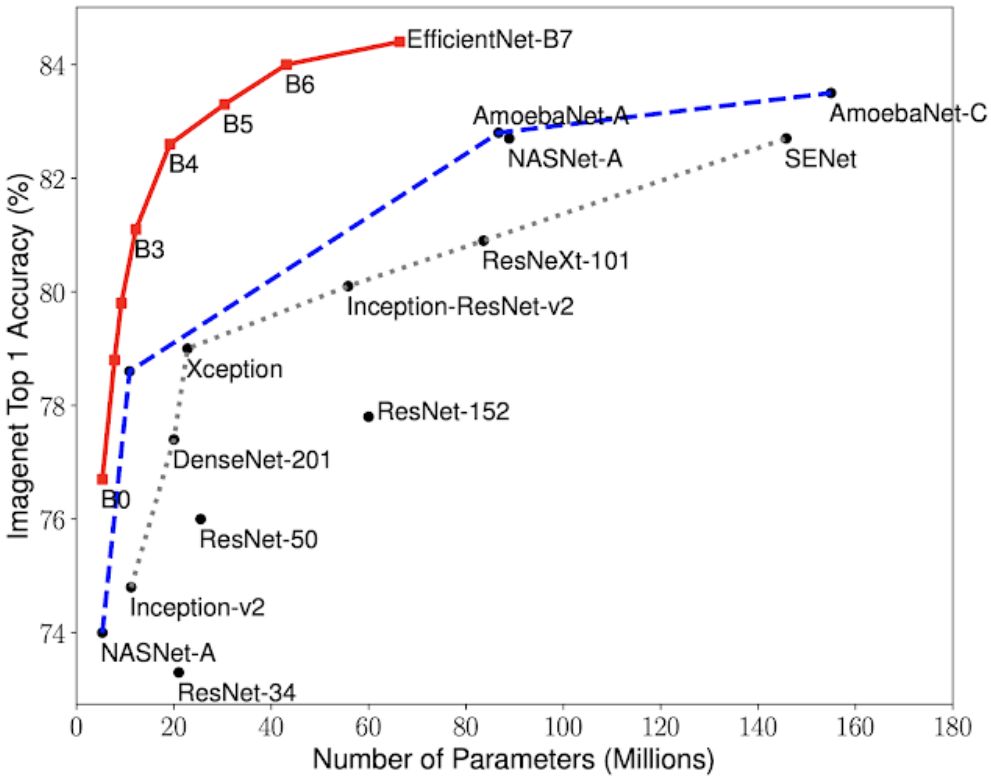


Рис. 3.14. Порівняння відношення точності до кількості параметрів з іншими нейромережами [63]

EfficientNetB7 був обраний для реалізації та дослідження, оскільки він показав найкращі результати у відношенні точності до кількості параметрів.

Висновки до розділу 3

В результаті виконаної роботи, описаної у цьому розділі можна підвести наступні підсумки:

1. Було визначено програмні та апаратні вимоги для реалізації спроектованого методу. Найбільш доцільним для виконання поставлених задач було обрано використання локального ПК, характеристики якого були приведені.
2. В якості програмного забезпечення середовищ для виконання та запуску програмного коду були обрані Anaconda та JupyterNotebook, оскільки вони повністю відповідають вимогам.
3. Для тренування нейромереж було досліджено та використано набір даних DIV2K, оскільки він є одним з найкращих в плані кількості та якості тестових зображень.
4. Для покращення існуючих методів збільшення реалістичності сприйняття, де використовується нейромережа VGG19, було досліджено та реалізовано нейромережі MobileNetV2 та EfficientNetB7, оскільки вони мають одні з найкращих показників у порівнянні з аналогами, які було описано у даному розділі.

РОЗДІЛ 4
РЕЗУЛЬТАТИ ТА АНАЛІЗ РЕАЛІЗОВАНОГО МЕТОДУ

Для об’єктивної оцінки в якості параметрів для порівняння реалізованого методу та базового, виступають якісні та кількісні значення, які ми отримали в процесі та результаті навчання та в результаті виконання роботи нейромереж. Кількісні параметри:

1. Втрата дискримінатора (Discriminator Loss) - найнижче значення коефіцієнта дискримінації для всіх етапів навчання.
2. Втрата сприйняття (Perceptual Loss) - найнижче значення коефіцієнта втрат сприйняття навчання для всіх етапів навчання.
3. Час на 1 крок навчання - середній час навчання 1 крок даної моделі в секундах.
4. Загальний час навчання - час навчання з конфігурацією, описаною нижче для спроектованої моделі.
5. Розмір моделі - скільки пам'яті займає конфігурація
6. Кількість параметрів нейромережі - сума всіх параметрів шару для даної моделі.

Якісні параметри:

1. Опрацьоване зображення з надвисокою роздільною здатністю використовуючи піксельну втрату (Pixel Loss)
2. Опрацьоване зображення з надвисокою роздільною здатністю використовуючи втрату сприйняття

4.1. Піксельна втрата

Піксельна втрата L^2(1) та втрата пікселів L^1 (2) часто використовуються у функціях втрат для навчання моделей надвисокої роздільної здатності. Вони вимірюють, відповідно, середню похибку квадратного пікселя та середню абсолютну похибку пікселів між зображенням HR I^{HR} та зображенням SR I^{SR}:

$$\alpha_{pixel,L^2}(I^{HR}, I^{SR}) = \frac{1}{HWC}||I^{HR} - I^{SR}||_2^2 \qquad (1)$$

$$\alpha_{pixel,L^1}(I^{HR}, I^{SR}) = \frac{1}{HWC}||I^{HR} - I^{SR}||_1^0 \qquad (2)$$

Де *H, W, C* - це висота, ширина та кількість каналів зображення відповідно. Pixel Loss L^2 безпосередньо оптимізує пікове відношення сигнал / шум (PSNR), показник оцінки, який часто використовується у порівнянні мереж SR.

Експерименти показали, що навіть краща продуктивність іноді досягається із втратою пікселів у L^1, і тому вона використовується для навчання EDSR та WDSR.

Основна проблема функцій піксельних втрат полягає в тому, що вони призводять до поганого сприйняття людиною. Сформовані зображення SR часто не мають високочастотного вмісту, реалістичних текстур і сприймаються як розмиті. Ця проблема вирішується за допомогою функцій втрати сприйняття (Perceptual Loss).

4.2. Втрата сприйняття

Важливою частиною збільшення роздільної здатності зображення є фотореалізм, для цього ми використовуємо зображення з надвисокою роздільною здатністю за допомогою змагальної генеруючої мережі (SRGAN). Автори використовують функцію сприйняття втрат, що складається із втрати вмісту та змагальності. Коли вміст втрачається, глибокі функції, витягнуті із зображень SR та HR, порівнюються із попередньо навченою нейронною мережею, у нашому випадку ми порівнюємо продуктивність цих мереж (3), яку ми позначимо як φ.

$$\alpha_{content}(I^{HR}, I^{SR}; \phi, l) = \frac{1}{H_l W_l C_l}||\phi_l(I^{HR}) - \phi_l(I^{SR}) \qquad (3)$$

Де φ - це карта об'єктів на рівні l, а H_l, W_l і C_l - це висота, ширина та кількість каналів цієї карти об'єктів відповідно. Вони також навчають свою модель надвисокої роздільної здатності як генератор *G* у генеруючій змагальній мережі (GAN). Дискримінатор GAN *D* оптимізований для вилучення SR із

зображень HR, тоді як генератор оптимізований для створення більш реалістичних зображень SR. Вони уніфікують втрати генератора (4).

$$\alpha_{generator}(I^{LR}; G, D) = -\log D(G(I^{LR})) \qquad (4)$$

З втратою вмісту перед втратою сприйняття, що використовується як ціль оптимізації для навчання моделі супер-роздільної здатності (5)

$$\alpha_{perceptual} = \alpha_{content} + 10^{-3}\alpha_{generator} \qquad (5)$$

Замість навчання моделі з надвисокою роздільною здатністю, тобто генератора з нуля в GAN, вони попередньо тренують її із втратою пікселів і налаштовують модель з втратами. Модель SRGAN використовує SRResNet як модель надвисокої чіткості, попередник EDSR.

4.3. Втрата дискримінатора

Функція збитків використовує стандартні втрати дискримінатора GAN для навчання втрат дискримінатора. Дискримінатор прагне максимізувати середнє значення логарифму ймовірності для реальних зображень та логарифму перевернутих ймовірностей фальшивих зображень (6).

$$max \sum logD(x) + log(1D(G(z)))eq \qquad (6)$$

$$min \sum y_{true} * -log(y_{pred}) + (1y_{true}) * -log(1y_{pred})eq \quad (7)$$

Якщо реалізувати безпосередньо, для цього потрібно буде змінити ваги моделі, використовуючи стохастичний підйом, а не стохастичний спуск. Це найчастіше реалізується як традиційна задача двійкової класифікації з позначками 0 та 1 для сформованих та реальних зображень відповідно. Модель покликана мінімізувати (7) середню двійкову перехресну ентропію, яку також називають логарифмічною втратою.

4.4. Результати

Для експериментів моделі EDSR та WDSR були точно налаштовані, оскільки SRGAN показує найкращі результати для цього підходу. Для отримання

загальних необхідних результатів кількість кроків для навчальних моделей була встановлена на 30000 (300 епох по 100 кроків) для EDSR та WDSR та 2000 для SRGAN, оскільки нам потрібно отримати результат, який вказує на ефективність моделі, швидкість його навчання та швидкість обробки зображень. Так як для повного навчання системи для всіх моделей потрібно 300 000 кроків, потрібно багато часу та ресурсів. Наше завдання - отримати кількісний результат виконання, і ми сподіваємось, що якість обробленого зображення буде знижено і не відображатиме кінцевий результат, який можна отримати після проходження необхідної кількості навчальних кроків.

Табл. 4.1.

Порівняння основних параметрів у процесі навчання мереж

	VGG19	MobileNetV2	EfficientNetB7
Розмір моделі	549 MB	14 MB	256 MB
Кількість параметрів	143,667,240	3,538,984	66,658,687
Час на 1 крок (с)	2.22	1.38	1.42
Загальний час (с)	4317	2815	2986
Perceptual loss	0.0844	0.0041	0.0007
Discriminator loss	0.1266	0.8976	1.2326

Як результат, ми можемо визначити, що, порівняно з іншими моделями, базова модель VGG19 має найбільші: розмір, кількість параметрів, час навчання на один крок і для всіх кроків загалом, а також має найбільше значення втрат сприйняття. Але в порівнянні з EfficientNet та MobileNet, VGG19 має найнижчі втрати дискримінатора, що свідчить про те, що модель має більшу ефективність у співвідношенні зображень LR та HR, що, в свою чергу, впливає на якість базової обробки зображень при збільшенні.

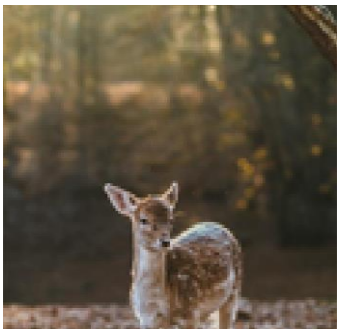
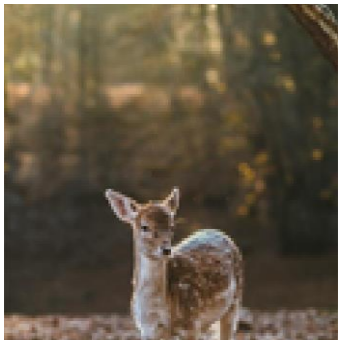
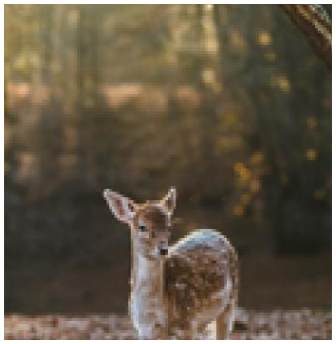
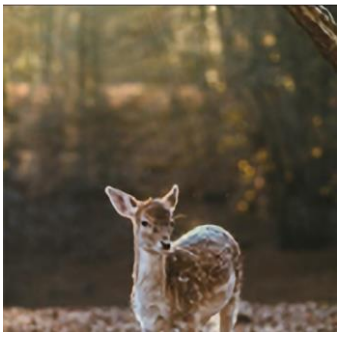
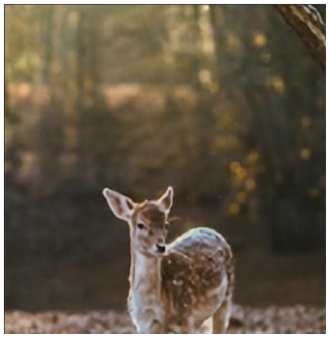
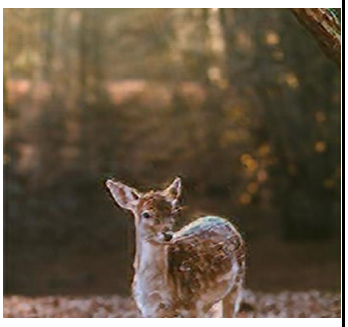
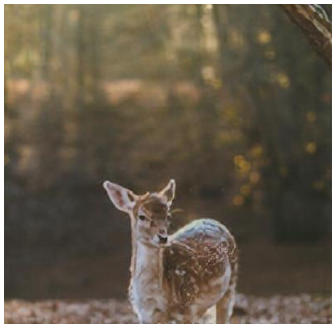
EfficientNet має приблизно половину розміру моделі та кількості параметрів у порівнянні з VGG19. Він також має середній час навчання та найбільше значення втрати дискримінатора, але в порівнянні з рештою, значення втрати сприйняття в цій мережі є найменшим, що свідчить про хорошу якість остаточної обробки зображення нейронною мережею, а саме його реалізм.

MobileNet показав себе з боку найлегшої моделі - він має найменший розмір і кількість параметрів, а також час на тренування, який для одного кроку, та і для всього навчання у цілому. Значення дискримінатора та сприйняття є середніми порівняно з іншими мережами, але тим не менше значення втрат сприйняття, порівняно з VGG19, все ще значно нижчі, як для MobileNet, так і для EfficientNet. Кількісні значення, основні показники точності втрат та час навчання для досліджуваних моделей представлені в таблиці 4.1.

Після тренування в подальших тестах використовували зображення з роздільною здатністю 124 x 118 пікселів, оскільки моделі тренувались за параметрами, що відповідають фрагментам зображення із заданою роздільною здатністю. Як результат, ми можемо оцінити результати виконання навчених мереж та їх ефективність у табл. 4.2.

Табл. 4.2.

Порівняння основних параметрів у результаті опрацювання зображення нейромережами

	EfficientNetB7	MobileNetV2	VGG19
Час опрацювання (с)	0.825	0.835	0.842
Input image			
SR (pixel loss)			
SR (perceptual loss)			

Висновки до розділу 4

В результаті даного розділу можна виділити наступні пункти:

1. Було визначено та описано параметри для порівняння нейромереж, та встановлено що вони є кількісними та якісними.
2. Було виконано навчання та отриманий результат роботи спроектованих та розроблених методів покращення зображення.
3. Отриманий результат було продемонстровано та виконано аналіз його параметрів.

Підсумовуючи значення, отримані в результаті навчання моделей, ми отримуємо відповідну якість обробки зображень. Таким чином, ми бачимо, що при обробці з втратою пікселів ми отримуємо приблизно однаковий візуальний результат, але оскільки MobileNet є найменш вимогливим до ресурсів, у цій категорії він має перевагу серед порівняних моделей, це демонструє час тренування для одного кроку, та усього тренування у цілому, а саме 1.38с та 2815с відповідно, коли у порівнюваних час відповідає 1.42с і 2986с для EfficientNet та 2.22с і 4317с для VGG19. Обробка з втратою сприйняття також показала відповідні результати, визначені під час тренування. Отже, у цій категорії EfficientNet має найкращі результати обробки, оскільки він має найнижчий середній коефіцієнт втрати сприйняття серед порівнянних моделей, а саме 0.0007 у порівнянні з 0.0041 у MobileNet та 0.0844 для VGG19.

Аналізуючи отримані результати у цілому, в даній роботі ми отримали метод, що має більш оптимізовані характеристики, але візуальна частина не дала необхідного результату, оскільки нейромережі, що використовувались не пройшли повний обсяг тренування, як було зазначено у базовому методі, а саме більше 200 000 кроків.

ВИСНОВКИ

В результаті даної роботи було визначено, що, моделі сприйняття, побудовані на нейронних мережах MobileNet та EfficientNet, виявилися швидшими в навчанні та мають кращий рівень втрат сприйняття, ніж VGG19, а це означає, що вони мають меншу потребу у обчислювальних ресурсах та можуть використовуватися на слабшому обладнанні. Це демонструють метрики часу для одного кроку навчання та усього навчання у цілому, а саме 1.38с та 2815с відповідно для MobileNet, 1.42с і 2986с для EfficientNet та 2.22с і 4317с для VGG19.

Відповідно для метрики середнього коефіцієнту втрати сприйняття ми отримали 0.0007 для EfficientNet у порівнянні з 0.0041 у MobileNet та 0.0844 для VGG19.

Оцінка якості результату не є об'єктивною, оскільки навчання було експериментальним та недостатнім, але тим не менше ми можемо спостерігати за результатами обробки зображень, що є вдосконалення, при повному тренуванні результат може бути покращений, таким чином ми програємо якість, але ми перемагаємо за продуктивністю. Де ми можемо застосувати це?

У сучасному світі телефони набувають все більшої популярності, а також використання потокових онлайн-сервісів. Таким чином, маючи хорошу продуктивність і легкість, при повному навчанні ми можемо модифікувати цей метод обробки зображень для обробки відеозображень, на виході ми отримуємо систему, яка може покращити якість відео на крайніх гаджетах, смартфонах, планшетних комп’ютерах і, відповідно, використовувати менше Інтернет-ресурсів при інтеграції з онлайн відео сервісами.

Оскільки час для обробки одного зображення вдається поступово зменшувати, а саме 0.825с для EfficientNet 0.835с для MobileNet та 0.842с для VGG19, то можна зробити висновок, що даний напрямок є перспективним і з'являються передумови для продовження досліджень в цьому напрямку для подальшого покращення аналогічної обробки відео.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Deep Learning for Image Super-resolution: A Survey [Електронний ресурс]/ Zhihao Wang, Jian Chen, Steven C.H., Hoi Fellow // IEEE 2020 – Режим доступу до ресурсу: https://arxiv.org/pdf/1902.06068.pdf
2. Super-resolution in medical imaging / H. Greenspan // The Computer Journal, vol. 52, 2008.
3. A super-resolution reconstruction algorithm for surveillance images / L. Zhang, H. Zhang, H. Shen, P. Li // Elsevier Signal Processing, vol. 90, 2010.
4. Is image superresolution helpful for other vision tasks / D. Dai, Y. Wang, Y. Chen, L. Van Gool // WACV, 2016.
5. Task-driven super resolution: Object detection in low-resolution images [Електронний ресурс] / M. Haris, G. Shakhnarovich, and N. Ukita // 2018 – Режим доступу до ресурсу: https://arxiv.org/pdf/1803.11316.pdf
6. Cubic convolution interpolation for digital image processing / R. Keys // IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 29, 1981.
7. Image and video upscaling from local self-examples / G. Freedman, R. Fattal // TOG, vol. 30, 2011.
8. Single-image super-resolution using sparse regression and natural image prior / K. I. Kim and Y. Kwon // TPAMI, vol. 32, 2010.
9. Super-resolution through neighbor embedding / H. Chang, D.-Y. Yeung, and Y. Xiong // CVPR, 2004.
10. Image superresolution as sparse representation of raw image patches / Y. Jianchao, J. Wright, T. Huang, Y. Ma // CVPR, 2008.
11. Learning a deep convolutional network for image super-resolution / C. Dong, C. C. Loy, K. He, X. Tang // ECCV, 2014.
12. Generative adversarial nets / I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. WardeFarley, S. Ozair, A. Courville, Y. Bengio // NIPS, 2014.
13. Photorealistic single image super-resolution using a generative adversarial network / A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang // CVPR, 2017.

14. Enhancenet: Single image super-resolution through automated texture synthesis / M. S. Sajjadi, B. Scholkopf, and M. Hirsch // ICCV, 2017.
15. Learning a single convolutional super-resolution network for multiple degradations / K. Zhang, W. Zuo, L. Zhang // CVPR, 2018.
16. Imagenet: A large-scale hierarchical image database / J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei // CVPR, 2009.
17. Microsoft coco: Common objects in context / T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, C. L. Zitnick // ECCV, 2014.
18. The pascal visual object classes challenge: A retrospective / M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman // IJCV, vol. 111, 2015.
19. Deep learning face attributes in the wild / Z. Liu, P. Luo, X. Wang, X. Tang // ICCV, 2015.
20. Memnet: A persistent memory network for image restoration / Y. Tai, J. Yang, X. Liu, C. Xu // ICCV, 2017.
21. Deep back-projection networks for super-resolution / M. Haris, G. Shakhnarovich, N. Ukita // CVPR, 2018.
22. Image quality assessment: From error visibility to structural similarity / Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli // IEEE Transactions on Image Processing, vol. 13, 2004.
23. A statistical evaluation of recent full reference image quality assessment algorithms / H. R. Sheikh, M. F. Sabir, A. C. Bovik // IEEE Transactions on Image Processing, vol. 15, 2006.
24. The unreasonable effectiveness of deep features as a perceptual metric / R. Zhang, P. Isola, A. A. Efros, E. Shechtman, O. Wang // CVPR, 2018.
25. Evaluation of image resolution and super-resolution on face recognition performance / C. Fookes, F. Lin, V. Chandran, S. Sridharan // Journal of Visual Communication and Image Representation, vol. 23, 2012.

26. Multi-scale structural similarity for image quality assessment / Z. Wang, E. Simoncelli, A. Bovik // Asilomar Conference on Signals, Systems, and Computers, 2003.
27. Fsim: a feature similarity index for image quality assessment / L. Zhang, L. Zhang, X. Mou, D. Zhang // IEEE transactions on Image Processing, vol. 20, 2011.
28. Making a completely blind image quality analyzer / A. Mittal, R. Soundararajan, A. C. Bovik // IEEE Signal Processing Letters, 2013.
29. Accelerating the superresolution convolutional neural network / C. Dong, C. C. Loy, X. Tang // ECCV, 2016.
30. Image super-resolution using deep convolutional networks // TPAMI, vol. 38, 2016.
31. Ntire 2017 challenge on single image super-resolution: Methods and results / R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, K. M. Lee // CVPRW, 2017.
32. Ntire 2017 challenge on single image super-resolution: Dataset and study / E. Agustsson and R. Timofte // CVPRW, 2017.
33. Pirm challenge on perceptual image enhancement on smartphones: report / A. Ignatov, R. Timofte, T. Van Vu, T. Minh Luu, T. X Pham, C. Van Nguyen, Y. Kim, J.-S. Choi, M. Kim, J. Huang // ECCV Workshop, 2018.
34. Accurate image superresolution using very deep convolutional networks / J. Kim, J. Kwon Lee, K. Mu Lee // CVPR, 2016.
35. Deep laplacian pyramid networks for fast and accurate superresolution / W.S. Lai, J.B. Huang, N. Ahuja, M.H. Yang // CVPR, 2017.
36. Fast and accurate image super-resolution with deep laplacian pyramid networks / W.S. Lai, J.B. Huang, N. Ahuja, M.H. Yang // TPAMI, 2018.
37. A fully progressive approach to single-image super-resolution / Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers // CVPRW, 2018.

38. Feedback network for image super-resolution / Z. Li, J. Yang, Z. Liu, X. Yang, G. Jeon, W. Wu // CVPR, 2019.
39. Recurrent backprojection network for video super-resolution / M. Haris, G. Shakhnarovich, and N. Ukita // CVPR, 2019.
40. Deep Residual Learning for Image Recognition [Електронний ресурс] / Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun // 2015 – Режим доступу до ресурсу: https://arxiv.org/pdf/1512.03385.pdf
41. Enhanced Deep Residual Networks for Single Image Super-Resolution [Електронний ресурс] / Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, Kyoung Mu Lee // 2017 – Режим доступу до ресурсу: https://arxiv.org/abs/1707.02921
42. Accurate image superresolution using very deep convolutional networks / J. Kim, J. Kwon Lee, K. M. Lee // CVPR 2016.
43. Wide Activation for Efficient and Accurate Image Super-Resolution [Електронний ресурс] / Jiahui Yu, Yuchen Fan, Jianchao Yang // 2018 – Режим доступу до ресурсу: https://arxiv.org/abs/1808.08718
44. Accurate image super-resolution using very deep convolutional networks / Jiwon Kim, Jung Kwon Lee, Kyoung Mu Lee // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 1646–1654.
45. Learning a deep convolutional network for image super-resolution / Chao Dong // European Conference on Computer Vision. Springer. 2014, pp. 184–199.
46. Accelerating the super-resolution convolutional neural network / Chao Dong, Chen Change Loy, Xiaoou Tang // European Conference on Computer Vision. Springer. 2016, pp. 391–407.
47. Real-time single image and video super-resolution using an efficient subpixel convolutional neural network / Wenzhe Shi // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 1874–1883.

48. Photo-realistic single image super-resolution using a generative adversarial network / Christian Ledig // IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. Vol. 2. 2018, p. 8.
49. Enhanced deep residual networks for single image super-resolution / Bee Lim // IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. Vol. 1. 2. 2017, p. 3.
50. Image Super-Resolution Using Dense Skip Connections / Tong Tong // IEEE (ICCV). 2017, pp. 4809–4817.
51. Residual Dense Network for Image Super-Resolution [Електронний ресурс] / Y. Zhang // 2018 – Режим доступу до ресурсу: https://arxiv.org/abs/1802.08797
52. Memnet: A persistent memory network for image restoration / Ying Tai // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017, pp. 4539–4547.
53. Deep residual learning for image recognition / Kaiming He // Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, pp. 770–778.
54. Aggregated residual transformations for deep neural networks / Saining Xie // Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on. IEEE. 2017, pp. 5987–5995.
55. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation [Електронний ресурс] / M. Sandler // 2018 – Режим доступу до ресурсу: https://arxiv.org/abs/1801.04381
56. The World's Most Popular Data Science Platform [Електронний ресурс] / Anaconda // Режим доступу до ресурсу: https://www.anaconda.com/
57. Jupiter Notebook [Електронний ресурс] // Режим доступу до ресурсу: https://jupyter.org/
58. Tensorflow [Електронний ресурс] Режим доступу до ресурсу: https://www.tensorflow.org/

59. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. IEEE Conference on Computer Vision and Pattern Recognition Workshops / Eirikur Agustsson, Radu Timofte // IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017.
60. Very Deep Convolutional Networks For Large-Scale Image Recognition [Електронний ресурс] / Karen Simonyan, Andrew Zisserman // 2015 – Режим доступу до ресурсу: https://arxiv.org/abs/1409.1556
61. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Електронний ресурс] / Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam // 2017 – Режим доступу до ресурсу: https://arxiv.org/abs/1704.04861
62. Image classification with MobileNet [Електронний ресурс] / Abhijeet Pujara // 2020 – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470
63. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks [Електронний ресурс] / Mingxing Tan, Quoc V. Le // 2020 – Режим доступу до ресурсу: https://arxiv.org/abs/1905.11946

ДОДАТОК А

Лістинг програми

Київ 2021

environment.yml

```
name: sisr
dependencies:
- python=3.6
- pip
- tqdm=4.32.1
- Pillow=6.0.0
- matplotlib=3.1.1
- pip:
  - tensorflow==2.3.1
  - tensorflow-addons==0.11.2
```

main.ipynb

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Add, Conv2D, Input, Lambda
from tensorflow.keras.models import Model

DIV2K_RGB_MEAN = np.array([0.4488, 0.4371, 0.4040]) * 255

def edsr(scale, num_filters=64, num_res_blocks=8, res_block_scaling=None):
    """Creates an EDSR model."""
    x_in = Input(shape=(None, None, 3))
    x = Lambda(normalize)(x_in)

    x = b = Conv2D(num_filters, 3, padding='same')(x)
    for i in range(num_res_blocks):
        b = res_block(b, num_filters, res_block_scaling)
    b = Conv2D(num_filters, 3, padding='same')(b)
    x = Add()([x, b])

    x = upsample(x, scale, num_filters)
    x = Conv2D(3, 3, padding='same')(x)

    x = Lambda(denormalize)(x)
    return Model(x_in, x, name="edsr")
def res_block(x_in, filters, scaling):
    """Creates an EDSR residual block."""
    x = Conv2D(filters, 3, padding='same', activation='relu')(x_in)
    x = Conv2D(filters, 3, padding='same')(x)
    if scaling:
        x = Lambda(lambda t: t * scaling)(x)
    x = Add()([x_in, x])
```

```
    return x
def upsample(x, scale, num_filters):
    def upsample_1(x, factor, **kwargs):
        """Sub-pixel convolution."""
        x = Conv2D(num_filters * (factor ** 2), 3, padding='same', **kwargs)(x)
        return Lambda(pixel_shuffle(scale=factor))(x)
    if scale == 2:
        x = upsample_1(x, 2, name='conv2d_1_scale_2')
    elif scale == 3:
        x = upsample_1(x, 3, name='conv2d_1_scale_3')
    elif scale == 4:
        x = upsample_1(x, 2, name='conv2d_1_scale_2')
        x = upsample_1(x, 2, name='conv2d_2_scale_2')
    return x
def pixel_shuffle(scale):
    return lambda x: tf.nn.depth_to_space(x, scale)
def normalize(x):
    return (x - DIV2K_RGB_MEAN) / 127.5
def denormalize(x):
    return x * 127.5 + DIV2K_RGB_MEAN

from data import DIV2K

train = DIV2K(scale=4, downgrade='bicubic', subset='train')
train_ds = train.dataset(batch_size=16, random_transform=True)
import os

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers.schedules import PiecewiseConstantDecay

weights_dir = 'weights/article'
os.makedirs(weights_dir, exist_ok=True)

model_edsr = edsr(scale=4, num_res_blocks=16)

optim_edsr = Adam(learning_rate=PiecewiseConstantDecay(boundaries=[20],
values=[1e-4, 5e-5]))

model_edsr.compile(optimizer=optim_edsr, loss='mean_absolute_error')
model_edsr.fit(train_ds, epochs=3, steps_per_epoch=10)

model_edsr.save_weights(os.path.join(weights_dir, 'weights-edsr-16-x4.h5'))

from model.wdsr import wdsr_b
```

```
model_wdsr = wdsr_b(scale=4, num_res_blocks=32)

optim_wdsr = Adam(learning_rate=PiecewiseConstantDecay(boundaries=[20],
values=[1e-3, 5e-4]))

model_wdsr.compile(optimizer=optim_wdsr, loss='mean_absolute_error')
model_wdsr.fit(train_ds, epochs=3, steps_per_epoch=10)

model_wdsr.save_weights(os.path.join(weights_dir, 'weights-wdsr-b-32-x4.h5'))

from model import srgan
import time

mean_squared_error = tf.keras.losses.MeanSquaredError()
binary_cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=False)

vgg = srgan.vgg_54()

generator = edsr(scale=4, num_res_blocks=16)
generator.load_weights(os.path.join(weights_dir, 'weights-edsr-16-x4.h5'))

discriminator = srgan.discriminator()

schedule = PiecewiseConstantDecay(boundaries=[1000], values=[1e-4, 1e-5])
generator_optimizer = Adam(learning_rate=schedule)
discriminator_optimizer = Adam(learning_rate=schedule)

def generator_loss(sr_out):
    return binary_cross_entropy(tf.ones_like(sr_out), sr_out)

def discriminator_loss(hr_out, sr_out):
    hr_loss = binary_cross_entropy(tf.ones_like(hr_out), hr_out)
    sr_loss = binary_cross_entropy(tf.zeros_like(sr_out), sr_out)
    return hr_loss + sr_loss

@tf.function
def content_loss(hr, sr):
    sr = tf.keras.applications.vgg19.preprocess_input(sr)
    hr = tf.keras.applications.vgg19.preprocess_input(hr)
    sr_features = vgg(sr) / 12.75
    hr_features = vgg(hr) / 12.75
    return mean_squared_error(hr_features, sr_features)
```

```
@tf.function
def train_step(lr, hr):

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        lr = tf.cast(lr, tf.float32)
        hr = tf.cast(hr, tf.float32)

        sr = generator(lr, training=True)
        hr_output = discriminator(hr, training=True)
        sr_output = discriminator(sr, training=True)

        # Compute losses
        con_loss = content_loss(hr, sr)
        gen_loss = generator_loss(sr_output)
        perc_loss = con_loss + 0.001 * gen_loss
        disc_loss = discriminator_loss(hr_output, sr_output)

    gradients_of_generator = gen_tape.gradient(perc_loss,
generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))

    return perc_loss, disc_loss

pls_metric = tf.keras.metrics.Mean()
dls_metric = tf.keras.metrics.Mean()

steps = 20
step = 0

for lr, hr in train_ds.take(steps):
    start = time.time()
    step += 1

    pl, dl = train_step(lr, hr)
    pls_metric(pl)
    dls_metric(dl)

    stop = time.time()
```

```
    duration = stop-start
    print(duration)
    if step % 10 == 0:
        print(f'{step}/{steps}, perceptual loss = {pls_metric.result():.4f},
discriminator loss = {dls_metric.result():.4f}')
        pls_metric.reset_states()
        dls_metric.reset_states()

generator.save_weights(os.path.join(weights_dir, 'weights-edsr-16-x4-fine-
tuned.h5'))

generator = wdsr_b(scale=4, num_res_blocks=32)
generator.load_weights(os.path.join(weights_dir, 'weights-wdsr-b-32-x4.h5'))

generator.save_weights(os.path.join(weights_dir, 'weights-wdsr-b-32-x4-fine-
tuned.h5'))

import os
import matplotlib.pyplot as plt

from model import resolve_single
from utils import load_image

%matplotlib inline

def resolve_and_plot(model_pre_trained, model_fine_tuned, lr_image_path):
    lr = load_image(lr_image_path)

    sr_pt = resolve_single(model_pre_trained, lr)
    sr_ft = resolve_single(model_fine_tuned, lr)

    plt.figure(figsize=(20, 20))

    model_name = model_pre_trained.name.upper()
    images = [lr, sr_pt, sr_ft]
    titles = ['LR', f'SR ({model_name}, pixel loss)', f'SR ({model_name},
perceptual loss)']
    positions = [1, 3, 4]

    for i, (image, title, position) in enumerate(zip(images, titles,
positions)):
        plt.subplot(2, 2, position)
        plt.imshow(image)
        plt.title(title)
```

```
        plt.xticks([])
        plt.yticks([])

weights_dir = 'weights/article'

start = time.time()
edsr_pre_trained = edsr(scale=4, num_res_blocks=16)
edsr_pre_trained.load_weights(os.path.join(weights_dir, 'weights-edsr-16-
x4.h5'))

edsr_fine_tuned = edsr(scale=4, num_res_blocks=16)
edsr_fine_tuned.load_weights(os.path.join(weights_dir, 'weights-edsr-16-x4-fine-
tuned.h5'))

resolve_and_plot(edsr_pre_trained, edsr_fine_tuned, 'demo/0001x4-crop.png')
stop = time.time()
duration = stop-start
print(duration)
```

data.py

```
import os
import tensorflow as tf

from tensorflow.python.data.experimental import AUTOTUNE


class DIV2K:
   def __init__(self,
                scale=2,
                subset='train',
                downgrade='bicubic',
                images_dir='.div2k/images',
                caches_dir='.div2k/caches'):

       self._ntire_2018 = True

       _scales = [2, 3, 4, 8]

       if scale in _scales:
           self.scale = scale
       else:
           raise ValueError(f'scale must be in ${_scales}')

       if subset == 'train':
```

```
            self.image_ids = range(1, 801)
        elif subset == 'valid':
            self.image_ids = range(801, 901)
        else:
            raise ValueError("subset must be 'train' or 'valid'")

        _downgrades_a = ['bicubic', 'unknown']
        _downgrades_b = ['mild', 'difficult']

        if scale == 8 and downgrade != 'bicubic':
            raise ValueError(f'scale 8 only allowed for bicubic downgrade')

        if downgrade in _downgrades_b and scale != 4:
            raise ValueError(f'{downgrade} downgrade requires scale 4')

        if downgrade == 'bicubic' and scale == 8:
            self.downgrade = 'x8'
        elif downgrade in _downgrades_b:
            self.downgrade = downgrade
        else:
            self.downgrade = downgrade
            self._ntire_2018 = False

        self.subset = subset
        self.images_dir = images_dir
        self.caches_dir = caches_dir

        os.makedirs(images_dir, exist_ok=True)
        os.makedirs(caches_dir, exist_ok=True)

    def __len__(self):
        return len(self.image_ids)

    def dataset(self, batch_size=16, repeat_count=None, random_transform=True):
        ds = tf.data.Dataset.zip((self.lr_dataset(), self.hr_dataset()))
        if random_transform:
            ds = ds.map(lambda lr, hr: random_crop(lr, hr, scale=self.scale),
num_parallel_calls=AUTOTUNE)
            ds = ds.map(random_rotate, num_parallel_calls=AUTOTUNE)
            ds = ds.map(random_flip, num_parallel_calls=AUTOTUNE)
        ds = ds.batch(batch_size)
        ds = ds.repeat(repeat_count)
        ds = ds.prefetch(buffer_size=AUTOTUNE)
        return ds
```

```
    def hr_dataset(self):
        if not os.path.exists(self._hr_images_dir()):
            download_archive(self._hr_images_archive(), self.images_dir,
extract=True)

        ds =
self._images_dataset(self._hr_image_files()).cache(self._hr_cache_file())

        if not os.path.exists(self._hr_cache_index()):
            self._populate_cache(ds, self._hr_cache_file())

        return ds

    def lr_dataset(self):
        if not os.path.exists(self._lr_images_dir()):
            download_archive(self._lr_images_archive(), self.images_dir,
extract=True)

        ds =
self._images_dataset(self._lr_image_files()).cache(self._lr_cache_file())

        if not os.path.exists(self._lr_cache_index()):
            self._populate_cache(ds, self._lr_cache_file())

        return ds

    def _hr_cache_file(self):
        return os.path.join(self.caches_dir, f'DIV2K_{self.subset}_HR.cache')

    def _lr_cache_file(self):
        return os.path.join(self.caches_dir,
f'DIV2K_{self.subset}_LR_{self.downgrade}_X{self.scale}.cache')

    def _hr_cache_index(self):
        return f'{self._hr_cache_file()}.index'

    def _lr_cache_index(self):
        return f'{self._lr_cache_file()}.index'

    def _hr_image_files(self):
        images_dir = self._hr_images_dir()
        return [os.path.join(images_dir, f'{image_id:04}.png') for image_id in
self.image_ids]
```

```
    def _lr_image_files(self):
        images_dir = self._lr_images_dir()
        return [os.path.join(images_dir, self._lr_image_file(image_id)) for
image_id in self.image_ids]

    def _lr_image_file(self, image_id):
        if not self._ntire_2018 or self.scale == 8:
            return f'{image_id:04}x{self.scale}.png'
        else:
            return f'{image_id:04}x{self.scale}{self.downgrade[0]}.png'

    def _hr_images_dir(self):
        return os.path.join(self.images_dir, f'DIV2K_{self.subset}_HR')

    def _lr_images_dir(self):
        if self._ntire_2018:
            return os.path.join(self.images_dir,
f'DIV2K_{self.subset}_LR_{self.downgrade}')
        else:
            return os.path.join(self.images_dir,
f'DIV2K_{self.subset}_LR_{self.downgrade}', f'X{self.scale}')

    def _hr_images_archive(self):
        return f'DIV2K_{self.subset}_HR.zip'

    def _lr_images_archive(self):
        if self._ntire_2018:
            return f'DIV2K_{self.subset}_LR_{self.downgrade}.zip'
        else:
            return f'DIV2K_{self.subset}_LR_{self.downgrade}_X{self.scale}.zip'

    @staticmethod
    def _images_dataset(image_files):
        ds = tf.data.Dataset.from_tensor_slices(image_files)
        ds = ds.map(tf.io.read_file)
        ds = ds.map(lambda x: tf.image.decode_png(x, channels=3),
num_parallel_calls=AUTOTUNE)
        return ds

    @staticmethod
    def _populate_cache(ds, cache_file):
        print(f'Caching decoded images in {cache_file} ...')
        for _ in ds: pass
```

```
        print(f'Cached decoded images in {cache_file}.')

def random_crop(lr_img, hr_img, hr_crop_size=96, scale=2):
    lr_crop_size = hr_crop_size // scale
    lr_img_shape = tf.shape(lr_img)[:2]

    lr_w = tf.random.uniform(shape=(), maxval=lr_img_shape[1] - lr_crop_size + 1,
dtype=tf.int32)
    lr_h = tf.random.uniform(shape=(), maxval=lr_img_shape[0] - lr_crop_size + 1,
dtype=tf.int32)

    hr_w = lr_w * scale
    hr_h = lr_h * scale

    lr_img_cropped = lr_img[lr_h:lr_h + lr_crop_size, lr_w:lr_w + lr_crop_size]
    hr_img_cropped = hr_img[hr_h:hr_h + hr_crop_size, hr_w:hr_w + hr_crop_size]

    return lr_img_cropped, hr_img_cropped


def random_flip(lr_img, hr_img):
    rn = tf.random.uniform(shape=(), maxval=1)
    return tf.cond(rn < 0.5,
                   lambda: (lr_img, hr_img),
                   lambda: (tf.image.flip_left_right(lr_img),
                            tf.image.flip_left_right(hr_img)))


def random_rotate(lr_img, hr_img):
    rn = tf.random.uniform(shape=(), maxval=4, dtype=tf.int32)
    return tf.image.rot90(lr_img, rn), tf.image.rot90(hr_img, rn)

def download_archive(file, target_dir, extract=True):
    source_url = f'http://data.vision.ee.ethz.ch/cvl/DIV2K/{file}'
    target_dir = os.path.abspath(target_dir)
    tf.keras.utils.get_file(file, source_url, cache_subdir=target_dir,
extract=extract)
    os.remove(os.path.join(target_dir, file))
```

train.py

```
import time
import tensorflow as tf

from model import evaluate
```

```
from model import srgan

from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.losses import MeanAbsoluteError
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import Mean
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers.schedules import PiecewiseConstantDecay


class Trainer:
   def __init__(self,
                model,
                loss,
                learning_rate,
                checkpoint_dir='./ckpt/edsr'):

       self.now = None
       self.loss = loss
       self.checkpoint = tf.train.Checkpoint(step=tf.Variable(0),
                                             psnr=tf.Variable(-1.0),
                                             optimizer=Adam(learning_rate),
                                             model=model)
       self.checkpoint_manager =
tf.train.CheckpointManager(checkpoint=self.checkpoint,

directory=checkpoint_dir,
                                                          max_to_keep=3)

       self.restore()

   @property
   def model(self):
       return self.checkpoint.model

   def train(self, train_dataset, valid_dataset, steps, evaluate_every=1000,
save_best_only=False):
       loss_mean = Mean()

       ckpt_mgr = self.checkpoint_manager
       ckpt = self.checkpoint

       self.now = time.perf_counter()
```

```
        for lr, hr in train_dataset.take(steps - ckpt.step.numpy()):
            ckpt.step.assign_add(1)
            step = ckpt.step.numpy()

            loss = self.train_step(lr, hr)
            loss_mean(loss)

            if step % evaluate_every == 0:
                loss_value = loss_mean.result()
                loss_mean.reset_states()

                # Compute PSNR on validation dataset
                psnr_value = self.evaluate(valid_dataset)

                duration = time.perf_counter() - self.now
                print(f'{step}/{steps}: loss = {loss_value.numpy():.3f}, PSNR =
{psnr_value.numpy():3f} ({duration:.2f}s)')

                if save_best_only and psnr_value <= ckpt.psnr:
                    self.now = time.perf_counter()
                    # skip saving checkpoint, no PSNR improvement
                    continue

                ckpt.psnr = psnr_value
                ckpt_mgr.save()

                self.now = time.perf_counter()

    @tf.function
    def train_step(self, lr, hr):
        with tf.GradientTape() as tape:
            lr = tf.cast(lr, tf.float32)
            hr = tf.cast(hr, tf.float32)

            sr = self.checkpoint.model(lr, training=True)
            loss_value = self.loss(hr, sr)

        gradients = tape.gradient(loss_value,
self.checkpoint.model.trainable_variables)
        self.checkpoint.optimizer.apply_gradients(zip(gradients,
self.checkpoint.model.trainable_variables))

        return loss_value
```

```
    def evaluate(self, dataset):
        return evaluate(self.checkpoint.model, dataset)

    def restore(self):
        if self.checkpoint_manager.latest_checkpoint:
            self.checkpoint.restore(self.checkpoint_manager.latest_checkpoint)
            print(f'Model restored from checkpoint at step
{self.checkpoint.step.numpy()}.')

class EdsrTrainer(Trainer):
    def __init__(self,
                 model,
                 checkpoint_dir,
                 learning_rate=PiecewiseConstantDecay(boundaries=[200000],
values=[1e-4, 5e-5])):
        super().__init__(model, loss=MeanAbsoluteError(),
learning_rate=learning_rate, checkpoint_dir=checkpoint_dir)

    def train(self, train_dataset, valid_dataset, steps=300000,
evaluate_every=1000, save_best_only=True):
        super().train(train_dataset, valid_dataset, steps, evaluate_every,
save_best_only)

class WdsrTrainer(Trainer):
    def __init__(self,
                 model,
                 checkpoint_dir,
                 learning_rate=PiecewiseConstantDecay(boundaries=[200000],
values=[1e-3, 5e-4])):
        super().__init__(model, loss=MeanAbsoluteError(),
learning_rate=learning_rate, checkpoint_dir=checkpoint_dir)

    def train(self, train_dataset, valid_dataset, steps=300000,
evaluate_every=1000, save_best_only=True):
        super().train(train_dataset, valid_dataset, steps, evaluate_every,
save_best_only)

class SrganGeneratorTrainer(Trainer):
    def __init__(self,
                 model,
                 checkpoint_dir,
                 learning_rate=1e-4):
```

```
        super().__init__(model, loss=MeanSquaredError(),
learning_rate=learning_rate, checkpoint_dir=checkpoint_dir)

    def train(self, train_dataset, valid_dataset, steps=1000000,
evaluate_every=1000, save_best_only=True):
        super().train(train_dataset, valid_dataset, steps, evaluate_every,
save_best_only)

class SrganTrainer:
    def __init__(self,
                 generator,
                 discriminator,
                 content_loss='VGG54',
                 learning_rate=PiecewiseConstantDecay(boundaries=[100000],
values=[1e-4, 1e-5])):

        if content_loss == 'VGG22':
            self.vgg = srgan.vgg_22()
        elif content_loss == 'VGG54':
            self.vgg = srgan.vgg_54()
        else:
            raise ValueError("content_loss must be either 'VGG22' or 'VGG54'")

        self.content_loss = content_loss
        self.generator = generator
        self.discriminator = discriminator
        self.generator_optimizer = Adam(learning_rate=learning_rate)
        self.discriminator_optimizer = Adam(learning_rate=learning_rate)

        self.binary_cross_entropy = BinaryCrossentropy(from_logits=False)
        self.mean_squared_error = MeanSquaredError()

    def train(self, train_dataset, steps=200000):
        pls_metric = Mean()
        dls_metric = Mean()
        step = 0

        for lr, hr in train_dataset.take(steps):
            step += 1

            pl, dl = self.train_step(lr, hr)
            pls_metric(pl)
            dls_metric(dl)
```

```
            if step % 50 == 0:
                print(f'{step}/{steps}, perceptual loss =
{pls_metric.result():.4f}, discriminator loss = {dls_metric.result():.4f}')
                pls_metric.reset_states()
                dls_metric.reset_states()

    @tf.function
    def train_step(self, lr, hr):
        with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
            lr = tf.cast(lr, tf.float32)
            hr = tf.cast(hr, tf.float32)

            sr = self.generator(lr, training=True)

            hr_output = self.discriminator(hr, training=True)
            sr_output = self.discriminator(sr, training=True)

            con_loss = self._content_loss(hr, sr)
            gen_loss = self._generator_loss(sr_output)
            perc_loss = con_loss + 0.001 * gen_loss
            disc_loss = self._discriminator_loss(hr_output, sr_output)

        gradients_of_generator = gen_tape.gradient(perc_loss,
self.generator.trainable_variables)
        gradients_of_discriminator = disc_tape.gradient(disc_loss,
self.discriminator.trainable_variables)

        self.generator_optimizer.apply_gradients(zip(gradients_of_generator,
self.generator.trainable_variables))

self.discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
self.discriminator.trainable_variables))

        return perc_loss, disc_loss

    @tf.function
    def _content_loss(self, hr, sr):
        sr = preprocess_input(sr)
        hr = preprocess_input(hr)
        sr_features = self.vgg(sr) / 12.75
        hr_features = self.vgg(hr) / 12.75
        return self.mean_squared_error(hr_features, sr_features)

    def _generator_loss(self, sr_out):
```

```
        return self.binary_cross_entropy(tf.ones_like(sr_out), sr_out)

    def _discriminator_loss(self, hr_out, sr_out):
        hr_loss = self.binary_cross_entropy(tf.ones_like(hr_out), hr_out)
        sr_loss = self.binary_cross_entropy(tf.zeros_like(sr_out), sr_out)
        return hr_loss + sr_loss
```

utils.py

```
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image

def load_image(path):
    return np.array(Image.open(path))
def plot_sample(lr, sr):
    plt.figure(figsize=(20, 10))
    images = [lr, sr]
    titles = ['LR', f'SR (x{sr.shape[0] // lr.shape[0]})']

    for i, (img, title) in enumerate(zip(images, titles)):
        plt.subplot(1, 2, i+1)
        plt.imshow(img)
        plt.title(title)
        plt.xticks([])
        plt.yticks([])
```

model/common.py

```
import numpy as np
import tensorflow as tf

DIV2K_RGB_MEAN = np.array([0.4488, 0.4371, 0.4040]) * 255

def resolve_single(model, lr):
    return resolve(model, tf.expand_dims(lr, axis=0))[0]

def resolve(model, lr_batch):
    lr_batch = tf.cast(lr_batch, tf.float32)
    sr_batch = model(lr_batch)
    sr_batch = tf.clip_by_value(sr_batch, 0, 255)
    sr_batch = tf.round(sr_batch)
    sr_batch = tf.cast(sr_batch, tf.uint8)
    return sr_batch
```

```
def evaluate(model, dataset):
    psnr_values = []
    for lr, hr in dataset:
        sr = resolve(model, lr)
        psnr_value = psnr(hr, sr)[0]
        psnr_values.append(psnr_value)
    return tf.reduce_mean(psnr_values)

def normalize(x, rgb_mean=DIV2K_RGB_MEAN):
    return (x - rgb_mean) / 127.5

def denormalize(x, rgb_mean=DIV2K_RGB_MEAN):
    return x * 127.5 + rgb_mean

def normalize_01(x):
    """Normalizes RGB images to [0, 1]."""
    return x / 255.0

def normalize_m11(x):
    """Normalizes RGB images to [-1, 1]."""
    return x / 127.5 - 1

def denormalize_m11(x):
    """Inverse of normalize_m11."""
    return (x + 1) * 127.5

def psnr(x1, x2):
    return tf.image.psnr(x1, x2, max_val=255)

def pixel_shuffle(scale):
    return lambda x: tf.nn.depth_to_space(x, scale)
```

model/edsr.py

```
from tensorflow.python.keras.layers import Add, Conv2D, Input, Lambda
from tensorflow.python.keras.models import Model
from model.common import normalize, denormalize, pixel_shuffle

def edsr(scale, num_filters=64, num_res_blocks=8, res_block_scaling=None):
    x_in = Input(shape=(None, None, 3))
    x = Lambda(normalize)(x_in)

    x = b = Conv2D(num_filters, 3, padding='same')(x)
    for i in range(num_res_blocks):
        b = res_block(b, num_filters, res_block_scaling)
```

```
    b = Conv2D(num_filters, 3, padding='same')(b)
    x = Add()([x, b])

    x = upsample(x, scale, num_filters)
    x = Conv2D(3, 3, padding='same')(x)

    x = Lambda(denormalize)(x)
    return Model(x_in, x, name="edsr")

def res_block(x_in, filters, scaling):
    x = Conv2D(filters, 3, padding='same', activation='relu')(x_in)
    x = Conv2D(filters, 3, padding='same')(x)
    if scaling:
        x = Lambda(lambda t: t * scaling)(x)
    x = Add()([x_in, x])
    return x

def upsample(x, scale, num_filters):
    def upsample_1(x, factor, **kwargs):
        x = Conv2D(num_filters * (factor ** 2), 3, padding='same', **kwargs)(x)
        return Lambda(pixel_shuffle(scale=factor))(x)

    if scale == 2:
        x = upsample_1(x, 2, name='conv2d_1_scale_2')
    elif scale == 3:
        x = upsample_1(x, 3, name='conv2d_1_scale_3')
    elif scale == 4:
        x = upsample_1(x, 2, name='conv2d_1_scale_2')
        x = upsample_1(x, 2, name='conv2d_2_scale_2')
    return x
```

model/wdsr.py

```
import tensorflow_addons as tfa
from tensorflow.python.keras.layers import Add, Conv2D, Input, Lambda
from tensorflow.python.keras.models import Model
from model.common import normalize, denormalize, pixel_shuffle

def wdsr_a(scale, num_filters=32, num_res_blocks=8, res_block_expansion=4,
res_block_scaling=None):
    return wdsr(scale, num_filters, num_res_blocks, res_block_expansion,
res_block_scaling, res_block_a)

def wdsr_b(scale, num_filters=32, num_res_blocks=8, res_block_expansion=6,
res_block_scaling=None):
```

```
    return wdsr(scale, num_filters, num_res_blocks, res_block_expansion,
res_block_scaling, res_block_b)

def wdsr(scale, num_filters, num_res_blocks, res_block_expansion,
res_block_scaling, res_block):
    x_in = Input(shape=(None, None, 3))
    x = Lambda(normalize)(x_in)

    # main branch
    m = conv2d_weightnorm(num_filters, 3, padding='same')(x)
    for i in range(num_res_blocks):
        m = res_block(m, num_filters, res_block_expansion, kernel_size=3,
scaling=res_block_scaling)
    m = conv2d_weightnorm(3 * scale ** 2, 3, padding='same',
name=f'conv2d_main_scale_{scale}')(m)
    m = Lambda(pixel_shuffle(scale))(m)

    # skip branch
    s = conv2d_weightnorm(3 * scale ** 2, 5, padding='same',
name=f'conv2d_skip_scale_{scale}')(x)
    s = Lambda(pixel_shuffle(scale))(s)

    x = Add()([m, s])
    x = Lambda(denormalize)(x)

    return Model(x_in, x, name="wdsr")

def res_block_a(x_in, num_filters, expansion, kernel_size, scaling):
    x = conv2d_weightnorm(num_filters * expansion, kernel_size, padding='same',
activation='relu')(x_in)
    x = conv2d_weightnorm(num_filters, kernel_size, padding='same')(x)
    if scaling:
        x = Lambda(lambda t: t * scaling)(x)
    x = Add()([x_in, x])
    return x


def res_block_b(x_in, num_filters, expansion, kernel_size, scaling):
    linear = 0.8
    x = conv2d_weightnorm(num_filters * expansion, 1, padding='same',
activation='relu')(x_in)
    x = conv2d_weightnorm(int(num_filters * linear), 1, padding='same')(x)
    x = conv2d_weightnorm(num_filters, kernel_size, padding='same')(x)
    if scaling:
```

```
        x = Lambda(lambda t: t * scaling)(x)
    x = Add()([x_in, x])
    return x

def conv2d_weightnorm(filters, kernel_size, padding='same', activation=None,
**kwargs):
    return tfa.layers.WeightNormalization(Conv2D(filters, kernel_size,
padding=padding, activation=activation, **kwargs), data_init=False)
```

model/srgan.py

```
from tensorflow.python.keras.layers import Add, BatchNormalization, Conv2D,
Dense, Flatten, Input, LeakyReLU, PReLU, Lambda
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.applications.mobilenet_v2 import MobileNetV2
from model.common import pixel_shuffle, normalize_01, normalize_m11,
denormalize_m11

LR_SIZE = 24
HR_SIZE = 96

def upsample(x_in, num_filters):
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x_in)
    x = Lambda(pixel_shuffle(scale=2))(x)
    return PReLU(shared_axes=[1, 2])(x)

def res_block(x_in, num_filters, momentum=0.8):
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x_in)
    x = BatchNormalization(momentum=momentum)(x)
    x = PReLU(shared_axes=[1, 2])(x)
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x)
    x = BatchNormalization(momentum=momentum)(x)
    x = Add()([x_in, x])
    return x

def sr_resnet(num_filters=64, num_res_blocks=16):
    x_in = Input(shape=(None, None, 3))
    x = Lambda(normalize_01)(x_in)

    x = Conv2D(num_filters, kernel_size=9, padding='same')(x)
    x = x_1 = PReLU(shared_axes=[1, 2])(x)

    for _ in range(num_res_blocks):
        x = res_block(x, num_filters)
```

```
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x)
    x = BatchNormalization()(x)
    x = Add()([x_1, x])
    x = upsample(x, num_filters * 4)
    x = upsample(x, num_filters * 4)
    x = Conv2D(3, kernel_size=9, padding='same', activation='tanh')(x)
    x = Lambda(denormalize_m11)(x)
    return Model(x_in, x)

generator = sr_resnet

def discriminator_block(x_in, num_filters, strides=1, batchnorm=True,
momentum=0.8):
    x = Conv2D(num_filters, kernel_size=3, strides=strides, padding='same')(x_in)
    if batchnorm:
        x = BatchNormalization(momentum=momentum)(x)
    return LeakyReLU(alpha=0.2)(x)

def discriminator(num_filters=64):
    x_in = Input(shape=(HR_SIZE, HR_SIZE, 3))
    x = Lambda(normalize_m11)(x_in)
    x = discriminator_block(x, num_filters, batchnorm=False)
    x = discriminator_block(x, num_filters, strides=2)
    x = discriminator_block(x, num_filters * 2)
    x = discriminator_block(x, num_filters * 2, strides=2)
    x = discriminator_block(x, num_filters * 4)
    x = discriminator_block(x, num_filters * 4, strides=2)
    x = discriminator_block(x, num_filters * 8)
    x = discriminator_block(x, num_filters * 8, strides=2)
    x = Flatten()(x)
    x = Dense(1024)(x)
    x = LeakyReLU(alpha=0.2)(x)
    x = Dense(1, activation='sigmoid')(x)

    return Model(x_in, x)
def mobileNet_22():
    return _mobileNet(5)
def mobileNet_54():
    return _mobileNet(20)
def _mobileNet(output_layer):
    mobileNet = MobileNetV2(input_shape=(None, None, 3), include_top=False)
    return Model(mobileNet.input, mobileNet.layers[output_layer].output)
```